Submitted to 1990 IEEE Intl. Conf. on Robotics and Automation

AD-A214 136

(1)

On The Motion of Compliantly-Connected Rigid Bodies in Contact, Part II: A System for Analyzing Designs for Assembly

Bruce R. Donald*

Dinesh K. Pai †

Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract:
The design of mechanical devices and the planning to assemble them should not be independent activities. ~~We introduce~~ a new, fully algorithmic, combinatorially precise approach to designing devices so that they are easy to assemble and (optionally) hard to disassemble. ~~Our~~ analysis can be used to validate good designs, and can be iterated to generate improved designs. The approach is based on an algorithm for predicting the motion of flexible objects in contact; the flexible objects have rotationally passively compliant members, which deform when they experience contact forces from the environment. Such objects are intended to model "snap-fastener"-type devices, which are very useful in design for assembly.

AD-A214 135

The algorithm is based on the theory in ~~our companion~~ paper [PD89]; in this paper ~~we~~ describe the details of the algorithm, its implementation in a system for predicting and analyzing the motion of snap-fastener-type devices, and experiments we ran using the system to analyze and design particular devices.

*This paper describes research done in the Computer Science Robotics Laboratory at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under grant No. IRI-8802390 and by a Presidential Young Investigator award, and in part by the Mathematical Sciences Institute.

†Supported in part by DARPA grant N0014-88-K-0591, MSI grant U03-8300, NSF grant DMC-86-17355, and by ONR grant N00014-86-K-0281.

89 10 30 214

To reduce our theory to practice, in this paper we focus on the following issues: the relevance of our approach to engineering (which we illustrate through the examples we ran using our system), the computational methods employed, the algebraic techniques for predicting motions in contact with rotational compliance, and issues of robustness and stability of our geometric and algebraic algorithms. Our computational viewpoint lies in the interface between differential theories of mechanics, and combinatorial collision detection algorithms. From this viewpoint, subtle mathematical difficulties arise in predicting motions under rotational compliance, such as the forced non-genericity of the intersection problems encountered in configuration space. We discuss these problems and their solutions. Finally, we extend our work to predict the forces on the manipulated objects as a function of time, and show how our algorithm can easily be extended to include uncertainty in control and initial conditions. With these extensions, we hope that our system could be used to analyze and design objects that are easy to assemble, even given control and sensing errors, and that require more force to disassemble than to mate.

1 Introduction

We are pursuing an algorithmic theory of *design for assembly*. To this end we are developing and implementing algorithms that can analyze and generate designs for objects so that they will be easy to assemble. In particular, we observe that real objects that robots might assemble are typically not rigid. For example, a Sony Walkman is made of plastic parts that snap together. Significant advances were made in the design of the IBM ProPrinter, by replacing traditional fasteners such as screws with plastic parts that simply snap together. The reason these plastic parts snap together is that they are *flexible*: more precisely, they are *passively compliant*. This means that when the parts are brought together and an external force applied, the parts deform in a prescribed way. More interestingly, the force required to mate two parts may be much less than the force required to take them apart.

Since we wish to be able to design and have our robots assemble such objects given task-level descriptions, we must have a systematic program for reasoning about and predicting their motions in contact. To this end, in our companion paper [PD89], we first made precise a sufficiently powerful notion of flexibility to model the objects above, which encompasses several important and complicated mechanisms in mechanical design and automated assembly: snap-fasteners, latches, ratchet and pawl mechanisms, and escapements. We modeled the physics of interaction between the flexible parts and the environment (including their mating parts). Then, using these tools, we proceeded to develop combinatorially precise algorithms for predicting the motion of a flexible object near and in contact with its mating part.

We have implemented our algorithm, and in this paper we describe the details of the algorithm, implementation, and experiments. We have built a system for predicting and analyzing the motion of "snap-fastener"-type devices, and we describe experiments we have run to analyze and design particular objects.

A major impediment to developing systems such as ours has been the apparent necessity to integrate out the differential mechanics in order to determine the long-term behavior of the system. This problem is exacerbated by the fact that in many models of rotational compliance such as the generalized damper (eg., [LMT, Erd, Don88a,b, Can89]), the resulting trajectories are not known to be algebraic; neither do we have ways of computing algebraic bounding approximations (or forward projections). We begin by discussing in quite general terms how our model of compliance permits us to obtain algebraic, closed-form solutions to motion prediction problems for a rotationally compliant object, and how this leads to *exact* algorithms for analyzing designs for assembly.

We continue by discussing the relevance of our approach to engineering. To this end, we give several examples of fastener-type objects. We illustrate our discussion with experiments performed with our system to automatically analyze and predict the motion during execution of the assembly plan.

Our algorithm is algebraic and exact in principle. However, unlike many theoretical algebraic algorithms, it is also implementable. A chief goal of this paper is to show how we implemented it. Next, we discuss our algorithm for motion analysis in detail. Our work is interdisciplinary, in that it is situated at the interface between differential theories of mechanics, and combinatorially precise computational approaches to collision detection and compliant motion prediction. We employ some simple tools from computational algebra, and we discuss their application in our algorithm in some detail. While these tools often seem straightforward from a theoretical viewpoint, there is a host of practical and implementational problems in trying to build a system and reduce them to practice. Many of these issues focus on the problems of robustness, non-genericity, and stability. We discuss these problems in some detail. For example, while many algebraic and computational-geometric algorithms can assume genericity (eg., general position), we can show that in the case of predicting rotational compliance, one is in effect forced to solve non-generic intersection problems in configuration space. Careful thought is required to make such algorithms robust.

Finally, we discuss two extensions to our algorithm which are needed in practice. Our algorithm, as formulated, assumes perfect control and perfect knowledge of the initial conditions; under these assumptions, it runs in time roughly $O(N^2 \log N)$ where N is the geometric complexity. However, real robots are subject to significant uncertainty in sensing and control. We show that our algorithm can be generalized to predict all possible outcomes of a motion, given uncertainty in the initial conditions, and in control, in time roughly $O(N^3 \log N)$. Moreover, the algorithm is still "exact" in the sense that it is a combinatorially precise decision procedure.

Our algorithm predicts where the motion will terminate, and the configuration and contact history as functions of time. We also show how our algorithm can predict the forces experienced by the manipulated parts as a function of time. Thus, we can avoid designs that require excessive forces to assemble, and can analyze how a design can be easier to mate than to disassemble. This force-history extension requires the introduction of transcendental functions, and so it results in a numerical (not an exact) algorithm; however, any desired precision for the forces may be obtained.[1]

This paper builds on [PD89], and so we assume the reader has a working knowledge of that paper.

2 Algebraic Solutions to Mechanics Prediction Problems

2.1 Problem Statement and Assumptions

Please refer to our companion paper [PD89] for a precise statement of the motion prediction problem and our assumptions of quasi-staticity, coulomb friction, etc. We refer to the static environment as "obstacles", although in fact, it will probably consist of

[1]By substituting a polynomial approximation for the transcendental functions, the algorithm could be made an ϵ-approximation scheme.

Figure 1: The root body is rectangular, and the motion plan is straight down. The two pawls are "hook-shaped", and are attached with torsional springs at the hinges to the root body. The environment ("obstacles") are the two filled-in rectangles meeting at right angles; they form a "T"-shaped black body. In this example, it seems that the pawls will clearly comply to the "T" and snap around it; presumably this device cannot be disassembled by reversing the assembly plan. In fact, our algorithm shows that (i) the right-hand pawl cannot reach the base of the "T", and (ii) with low friction, there is no obstacle to disassembling the device.

Figure 2: The result of running our algorithm on the two-pawl example. Notice how the pawls comply around the obstacles as they deform in response to the kinematic constraints and the contact forces. Figs. 2a-g show the motion downwards. At 2g, the root collides with the "T", and the motion is reversed. Figs. 2h-o show the reverse motion upwards.

fixtures and/or the mating half of the manipulated part. As in [PD89], we will consider the problem of moving a flexible, linked body $\mathcal{M}$ in the plane, in a polygonal environment $\mathcal{N}$. See fig 1. The motion "plan" to assemble the parts consists of a straight-line translation of the root body; this motion is parameterized by time and is specified by an initial configuration and a velocity vector. (Throughout this paper, we will use the term "assembly/disassembly plan" to mean such a straight-line translation). As the pawls contact the environment, they can deform in a prescribed way from the contact forces. In particular, each pawl is attached to the root by a torsional spring. The pawls can also "snap" off the obstacle edge back to their zero position; this motion is modeled using a pure rotation. As the motion proceeds, the pawls deform (deflect) around the obstacles in response to the kinematic constraints and the contact forces. See fig 2.

Fig. 2 was generated by our algorithm. Kinematic constraints are modeled using configuration space C-surfaces, as in [LoP, Don87]. Reaction forces are modeled using Coulomb friction. The dynamics of the system are assumed to be quasi-static [Whi82, Mas86, Pai88]; we regard this assumption as a 0^{th}-order approximation of the real dynamical system.

2.2 Computing Motions and Intersection Problems

We now provide a somewhat abstract view of the results of [PD89]. Here are the qualitative states of the root body: it is either undergoing a pure translation, or else it it is stuck due to incompatible kinematic constraints. It is clear [PD89] that for the root body, all we need to compute is the time at which this sticking occurs. This time is an upper bound on the simulation.

The state of a pawl is more complicated. When a pawl is in free-space undergoing a pure translation, it can strike a surface. This requires a translational collision detection algorithm. When the pawl is in contact with a surface, as time increases, its configuration (orientation in this case) must change so as to comply with this constraint. (We use the term "constraint" as in [LoP, Don87, PD89], to refer to the kinematic constraint that (a) an edge of the pawl touch a vertex of the environment or (b) a vertex of the pawl touch an edge of the environment. These constraints form type "(A)" and "(B)" configuration space surfaces, respectively). See, for example, fig. 2. In this case, the configuration space of the flexible object is $\Re^2 \times S^1 \times S^1$; a point in $\Re^2$ specifies the configuration of the root body, whereas each angle in S^1 specifies the orientation of the right and left pawls. In this case, we require an algorithm that can return the mapping from time to configuration, and the curve in configuration space which is the image of this mapping.

Now, as the pawl traces out this curve, three things can happen. First, the pawl may break contact with the surface constraint, due to incompatible kinematics. Second, the pawl may stick on the surface, due to force-balance from the reaction forces. (More generally, for every constraint, there may be at most two disjoint time intervals during which sliding (resp. sticking) occurs, separated by an interval during with sticking (resp sliding) occurs). Third, the pawl may strike another constraint. In [PD89], we perform the kinematic and physical analysis required to compute the times at which these events occur, and we described the algorithms. In particular, we showed how, given a constraint, to compute a quadratic function of time whose zeros define the endpoints of these intervals of sliding and sticking behavior.

Finally, when a pawl breaks contact and "snaps off", we must perform a pure rotational intersection test to determine where it snaps to. When a pawl lies on the intersection of two constraints, we must determine whether sticking occurs there due to incompatible constraints, or which constraint takes precedence and becomes a new constraint at that time. Again, see [PD89] for the analysis and algorithms for these computations.

2.3 Differential Theories of Mechanics

We view the motion prediction problem, even with rotational compliance and quasi-static mechanics, as a problem that can be solved by careful reduction to the intersection, or collision detection problems [Can86, Don84,87] that have received much attention. Our approach [PD89] to modeling rotational compliance and to incorporating frictional constraints leads to the first formulation of the motion prediction problem which permits a *reduction* of motion prediction to collision detection. Our solution differs from previous work on predicting, bounding, and planing rotationally compliant motions with quasi-static mechanics in that it is (i) *purely algebraic*, and hence exact, (ii) *combinatorially precise*, in that the computational complexity is exactly known, (iii) practical and implementable, and (iv) requires no integration. In this subsection we elaborate somewhat on these characteristics.

In order to predict the motion of objects in a mechanical system, we require a computational theory of mechanics. A *differential* theory of mechanics takes the instantaneous state and forces, and computes the resultant instantaneous motion.

Quasi-static analysis is another differential theory of mechanics, in that it predicts the instantaneous motion given state and friction forces.[2] In general, differential theories can be integrated, in the sense that the long-term behavior can be integrated out given the differential mechanics. We say that a theory is *closed-form integrable* when we can solve for the resultant curves in state-space in closed form. When a theory is closed-form integrable, we can develop exact, combinatorially precise algorithms for predicting and planning the motion of objects [Don88a,b, Can89, Briggs89]. When a closed-form solution is not known, numerical methods can be used to integrate out solutions in some cases.

Quasi-static mechanics and generalized damper dynamics are indeed closed form theories in the case of pure translation. However once rotational compliance—the tendency of a manipulated object to change its orientation in response to reaction forces from the environment—is introduced, we do not have closed-form integrable theories of mechanics. We do have precise, algorithmic theories of the differential mechanics, such as Erdmann's generalized friction cone [Erd84] and the acceleration center force-dual model of Brost and Mason [BM89], but these theories must be numerically integrated to obtain solutions. The best algorithms for this process are not combinatorially precise, and do not have solution accuracy bounds. This problem is exacerbated by uncertainty in sensing and control; in this case we are left with the very hard problem of trying to integrate out a stochastic vector field on a holonomic constraint, subject to a family of initial conditions.

In contrast, in [PD89] we use a well-known theory of mechanics with rotational compliance that is not only closed form integrable, but closed form *algebraic*. By this we *mean that the following*. Our theory is based on the differential mechanics, but can be integrated to produce solutions that are algebraic curves in the configuration space. In fact, the solutions paths are parameterized by time and are piecewise quadratic or linear. Since we can represent configuration space C-surfaces as quadric surfaces, all of the intersection problems described above (2.2) are no harder than intersecting a quadratic path with a quadric surface. Somewhat surprisingly, even the functions that determine at what times sliding and sticking will occur on a surface are also quadratic in the time parameter.

Closed form, algebraic solutions permit us to construct exact, combinatorially precise algorithms for the motion prediction problem. Furthermore, they allow us to straightforwardly generalize our techniques to encompass certain simple types of uncertainty in control and initial conditions.

In principle, our algorithms can be implemented using exact-precision, algebraic numbers. In practice, we use finite-precision approximation techniques. Robustness is a key issue, and algorithms that are theoretically correct with exact precision are often numerically unstable. A major component of our research consists of building a system that can strengthen the theoretical algorithms (eg, by adding consistency checks) to make them

[2]In the case of ambiguity or non-determinism, we view such a theory as a relation.

practical.

3 Relevance to Engineering and Design for Assembly

3.1 The Two-Pawl Example

Figure 2 shows motion predicted for a typical pawl-like device around a mating part (a black, "T"-shaped object). Now, for a real device, the pawls will be tiny in comparison to the root body and mating parts, but we have made them very large here so as to illustrate the geometric interaction. In this example, the coefficient of friction is very small. This enables the reverse motion to "pull the pawl back up" out of its mating part. The "assembly plan" is to move straight down. When a root collision is detected, the system attempts to reverse the assembly plan and pull straight up. Because there is no friction, the pawl can be removed. If the coefficient of friction is increased, then the left pawl will stick on left of the "T" environment, and the flexible object cannot be pulled back up. This example shows that while it is relatively clear, intuitively, that during assembly the pawls may snap around the black "T"-shaped object, that even this simple problem holds some surprises. First, the right-hand pawl cannot actually reach around the base of the "T" before the root collision occurs. Second, it is not *a priori* clear that the left pawl will not stick during the reverse (disassembly) plan. In fact, it will not stick on the left of the "T" merely due to kinematic constraints; friction is required.

3.2 "Motion Diode" Example

In design for assembly, we often desire "locking" parts that, when mated, cannot be disassembled by motion plans in a particular family of directions. More generally, we may require interlocking parts that cannot be disassembled at all, for any translational motion plan. Most generally, one might want parts that cannot be disassembled without exerting large forces (see sec. 4). Following a suggestion of Mason [Ma84], we call such objects "motion diodes." The term is motivated by the fact that motion is possible in certain directions, but not in others. Our usage differs from Mason's, in that his motion diodes are geometries from which a robot cannot be *guaranteed* to emerge. Our motion diodes are (flexible object, environment) pairs such that for some family of controls, (or perhaps all controls), no change in the sign of the controls can reverse the motion to reachieve the start position. In our simple case a "plan" is a translation given as a straight line motion

$$p = p_o + \dot{p}t \tag{1}$$

where t is the time, p_o is the initial position (at $t = 0$), and $\dot{p}$ is the root velocity. A family of controls corresponds to a set of velocities $\{\dot{p}\}$, and changing the sign amounts to specifying $-\dot{p}$.

Figure 3: "Motion diode" example. There is no root body, and only one triangular pawl which can pivot about a torsional spring at its center. Pure translations of the diode can be commanded, and the triangular pawl deflects in response to contact forces from the environment. These figures were generated by running our analysis algorithm on the data shown. In 3c, the pawl snaps off the upper right obstacle and continues downward. Fig. 4 shows the reverse motion, during which the pawl gets stuck comping back up.

Figure 4: "Motion diode" example. When we try to pull the flexible object back up (move in direction $+y$), it gets stuck due to kinematic constraints.

If motion diodes can be designed, analyzed and verified, then they can be rigidly attached as "fasteners" to bodies that we wish to mate, but not to disassemble. For example, if the triangular pawl in fig. 3 is attached in the z-axis (perpendicular to the figure) to a root body in a parallel x-y plane to the figure, then the root body can be fastened irreversibly to its mating part.

Our algorithm can analyze designs for these kinds of diodes. In fig. 2, the two-pawl device and the T-shaped object would form a motion diode with respect to pure translation in y, for sufficiently high coefficients of friction.

Now see fig. 3. In this example, there is no root body. The one triangular pawl can pivot about its center; a torsional spring is attached at the pivot. The pawl is moved down in a pure $-y$ translation, and in response to the reaction forces from the environment, it rotates compliantly. Let us label the black obstacles, starting with the uppermost one, in clockwise order, A, B, and C. The pawl contacts A and rotates counterclockwise while sliding along A's upper left corner. Eventually, the pawl breaks contact with A, and snaps off, only to hit the rightmost vertex of C. It briefly slides (while rotating compliantly) along C, until it hits B. The tighter constraint from B takes over, and the pawl is again "cocked" counterclockwise until it breaks contact at the lower left vertex of B. Finally, the pawl snaps off B to its rest position.

Now, this mechanical system is a "diode" with respect to pure y-translation (see the figures)—when the pawl is moved back up in the $+y$ direction, it jams due to incompatible kinematic constraints. More interestingly, if B and C are extended to the right and left (resp.), the system is a diode with respect to *all* translational motions. That is, no commanded translation can bring the flexible body back out of the hole between B and C. Our algorithm can decide that for a particular motion plan, a system is a diode. There also exists a theoretical extension of our algorithm, using the theory of real closed fields, which can decide whether the system is a diode with respect to *every* disassembly plan, but this algorithm is not practical. In practice, applying the algorithm to a discretization of the control set would be more practical, although not exact.

3.2.1 A Classification of Motion Diodes

Figure 5: In (a), we see a pawl-type mechanism affixed to a root body which is translating down in the $-y$ direction. The pawl is very similar to fig. 1. (b) forms a relative motion kinematic diode. Without constraint (c), there exist other translations that disassemble the "snapped onto" final configuration. If we make the distance *top* long enough so that the root collides with the top of *b*, then the addition of constraint (c) turns (b) into a total-motion kinematic diode. Now, (d) forms a friction diode, and (e) forms a force diode. The presence or absence of (c) determines whether (d) and (e) are total or relative motion diodes.

Consider figure 5, which is a gedankenexperiment illustrating different types of diodes. The flexible mechanism, and in particular the pawl structure is very similar to fig. 1. The commanded motion is in the $-y$ direction. Now, for geometry (b), the pawl deflects clockwise, and snaps onto the sharp bump at the lower left of (b). The motion is not reversible, due to kinematic constraints alone. Now, unless constraint (c) is present, there may exist other translations which will disassemble the mechanism. The addition of constraint (c) means that no translation can disassemble the mechanism (so long as (b) is "tall enough"—see fig. 5). Now, consider 5d. This geometry also prevents $+y$ motion, but the sticking is due to friction, not kinematics. Finally, consider 5e. Here, relatively small forces are required on the root in the $-y$ direction to assemble the object; however, large forces are required on the root in the $+y$ direction to disassemble the object. (e) is an object that it is easier to assemble than it is to disassemble.

One application of our algorithm has been to determine whether a system is a motion diode, and what kind of diode it is. Flexible mechanisms which function as motion diodes can be used to fasten one part to another, and to make the mechanical *connection robust* with respect to attempted relative motion of the two parts. This kind of analysis could be very useful in design for assembly. Using our algorithm, we can form the following *classification* of flexible mechanism motion diodes. The classification is "two dimensional" in the sense that one "axis" is relative vs. total motion, and the other axis is Kinematic vs friction vs force diodes.

More specifically, the first classification is

1. A *relative motion diode* is a (flexible object, environment) pair such that for some family of controls, called the *relative motions*, no change in the sign of the controls can reverse the motion to reachieve the start position.

2. A *total motion diode* is a relative motion diode for all relative motions.

The second classification is:

A A *kinematic* diode prevents reverse motion due to purely kinematic constraints.

B A *friction* diode prevents reverse motion due to a combination of kinematic constraints and coulomb friction. It depends on the coefficient of friction.

C A *force* diode is described in sec. 4. Essentially, one imagines replacing the control of a root body by generalized spring position control, thus equating displacements of and forces on the root body in a first-order relation. This permits one to ask *What forces are experienced by the pawls as a function of time?* and *What force is exerted on the root to cause the motion?* A force diode, roughly speaking, is a friction diode for all control forces below a certain modulus bound.

Hence, we see that without friction, figs. 1-2 is not a diode. With sufficiently high friction, it is a relative motion friction diode. Fig. 3 is a total motion kinematic diode. Our algorithm can decide all these classifications automatically.

In figure 5, we see that (b) is a kinematic diode, (d) is a friction diode, and (e) is a force diode. If we add constraint 5c and make (b), (d), and (e) "tall enough", then these are total motion diodes; otherwise they are relative.

One goal of our algorithm has been to provide an algorithmic means of classifying flexible objects such as snap-fasteners by diode type. Obviously, this is just a start, and other, finer types of classifications are possible. Ours seems useful in design, and is efficiently computable by implemented, precise algorithms.

4 Computing Mating Force Information

Consider figure 5e once again. We wish to make precise the notion that this is a force diode.

Force diodes are interesting and useful, in that we can use them to build objects that are easier to mate than to take apart. More generally, we wish to compute the forces required to assemble and disassemble our parts, because we must determine that excessive forces that could damage the parts are never exerted. Conversely, if an assembly we have designed can be taken apart with very small forces or by a wide range of motions, then the mechanical connection between the parts may be be insufficient. By extending our algorithm to calculate the forces the robot is required to exert (external forces) and the forces the parts experience while mating, we can develop a general tool that performs all these computations.

The forces and torques experienced by the root as result of the interaction of the pawls with the obstacles can be computed by considering the force balances on the pawls. We shall show the computation of the force and torque at the hinge point of a single pawl due to its contact. This information is useful to ascertain that the pawl does not experience excessive loading during assembly. It is straightforward to transform this force and torque into some other coordinate frame on the root body. The total force and torque on the root is found by summing the contributions of the individual pawls. The total force and torque information is important since it has to be supplied by the robot or assembly machine.

First of all, we observe that if the pawl sticks due to friction (as in the case of a friction diode) or due to incompatible kinematic constraints (as in the case of a kinematic diode), the forces will be infinite as time is increased. This is due to the assumption that the root and pawl are rigid bodies, but even in practice, we expect the forces to get extremely large. Hence, we need only to examine the case of the pawl sliding on the obstacles. Second, the results of [PD89] provides us with a mapping from time to configurations, i.e., to the angles of the pawl. Hence we shall derive the dependence of force on the configuration θ instead of on time.

We first consider the type-B contact: The notation is the same as that of [PD89]. The torque τ at the hinge point is only due to the displacement of the pawl from its resting configuration, θ_0. Hence,

$$\tau = k(\theta - \theta_0), \tag{2}$$

where k is the torsional stiffness of the spring connecting the pawl to the root. Now the force on the contact point during quasi-static sliding is some multiple of the vector $\boldsymbol{f}_e$, i.e., $\boldsymbol{f} = \alpha \boldsymbol{f}_e$. ($\boldsymbol{f}_e$ is the vector along the friction cone edge in the oposite direction of sliding; see [PD89]). Therefore, balancing the torques on the pawl about the hinge point,

$$\tau = \boldsymbol{r} \times \boldsymbol{f} = k(\theta - \theta_0). \tag{3}$$

Simplifying and rearranging terms, we can express α as a function of only θ:

$$\alpha = \frac{k(\theta - \theta_0)}{R_\theta \boldsymbol{p}_i \times (\boldsymbol{n}_j - \mu \boldsymbol{v})}. \tag{4}$$

The type-A case is similar. As in [PD89], we redefine $\boldsymbol{r}$ to be the vector from the hinge point to the contact vertex, $\boldsymbol{r} = \boldsymbol{p}_j - \boldsymbol{p}$, and $\boldsymbol{f}_e = R_\theta \boldsymbol{n}_i - \mu \boldsymbol{v}_\theta$. We have written the vector in the sliding direction as $\boldsymbol{v}_\theta$ instead of the usual $\boldsymbol{v}$ to emphasize that the direction depends on θ; more precisely, $\boldsymbol{v}_\theta = R_{\theta \pm \frac{\pi}{2}} \boldsymbol{n}_i$. Then performing a torque balance as in the type-B case, we get

$$\tau = \boldsymbol{r} \times -\boldsymbol{f} = k(\theta - \theta_0). \tag{5}$$

The negative sign appears in front of $\boldsymbol{f}$ since it was defined in [PD89] as acting on the obstacle. Simplifying and rearranging,

$$\alpha = \frac{k(\theta - \theta_0)}{(R_\theta \boldsymbol{n}_i - \mu \boldsymbol{v}_\theta) \times (\boldsymbol{p}_j - \boldsymbol{p})}. \tag{6}$$

The above expressions could be used directly to numerically compute the maximum force during sliding, using well known one-dimensional optimization methods. However, the maximum force and torque computation for a *single* pawl turns out to be much simpler. We define a *sliding segment* as a connected interval of time in which the pawl is sliding along an obstacle feature, without changing the contact topology. Since the torque τ varies linearly with θ, its maximum clearly occurs at a boundary of a sliding segment. We shall show below that the force maximum in a sliding segment can also occur only at a boundary of the segment. Hence one need only check the force and torque at these transition points. We have thus reduced the apparently continuous problem of finding the maxima to a discrete one.

Proposition 4.1 *Let*

$$\alpha(\psi) = \frac{A\psi + B}{C \sin\psi}, \tag{7}$$

Subject to

$$\alpha(\psi) > 0 \text{ for all } \psi \in I = [\psi_{min}, \psi_{max}]. \tag{8}$$

Then α does not achieve a relative maximum (and hence a global maximum) in the interior of I.

Proof. We look at the relative extrema of α in I and show that they are minima. Differentiating with respect to ψ,

$$\alpha' = \frac{\sin\psi A - (A\psi + B)\cos\psi}{C \sin^2\psi} \tag{9}$$

At a relative extremum, $\alpha' = 0$; hence the numerator of Equation 9 must be zero. Computing the second derivative of α at the extremum, and simplifying, we find

$$\alpha'' = \frac{A\psi + B}{C\sin\psi} = \alpha > 0. \tag{10}$$

Hence, the extremum is a relative minimum. □

Our claim follows by showing that our expressions for α are of the form required by the proposition. First, notice that for both Equations 4 and 6, the denominators simplify to the form $C_1 \sin\theta + C_2 \cos\theta$, where C_1 and C_2 are independent of θ, which can be easily rewritten as $C\sin(\theta + \phi)$, where $C = \sqrt{C_1^2 + C_2^2}$ and $\phi = \tan^{-1}\frac{C_2}{C_1}$. Hence, by making the substitution $\psi = \theta + \phi$, the expressions for α have the desired form for both type-B and type-A contacts. Finally, the requirement that $\alpha > 0$ in the sliding segment is nothing but the requirement that to maintain contact, the force has to be a positive multiple of f_e (see [PD89]). Hence the above proposition applies, and the maximum α and hence the maximum force, can only occur at the boundaries of a sliding segment.

With this extension to our algorithm, our system can compute the force required in assembly and disassembly, and also the forces experienced by the pawls during execution. However, the computation of forces is not exact (because they involve transcendental functions that cannot be "rationalized" as we do in sec. 5 for pure kinematic constraints to make them algebraic); and hence this extension makes our algorithm approximate. (The use of provably good polynomial approximations to the transcendental functions will make the algorithm a provably good approximation algorithm, however). The computation of mating forces is perhaps one of the most important aspects of our algorithm for design for assembly.

5 Collision Detection Algorithms and Intersection Problems

From an algorithmic point of view, much of our work has been to reduce the motion prediction problem to a series of collision detection problems. (More accurately, these problems are intersection problems in Cspace (configuration space), in the sense of [Don 84,85]; eg., one wishes to intersect a path with a C-surface, or with two Csurfaces, etc). Put bluntly, in this paper and especially in [PD89], we reduced the flexible object motion prediction problem to intersection problems (like collision detection) in Cspace. Now, we show how to reduce these intersection problems to problems in elementary elimination theory (particularly, the simultaneous solution of polynomial equations and inequalities). We go into some detail about how to solve such systems as specialized to our particular application, and how to implement the solution. We have two goals. The first is to elucidate a theoretical, exact, algebraic algorithm that is correct. However, this algorithm is numerically unstable when implemented with finite precision arithmetic. Hence robustness is a key issue. We emphasize the steps we have taken in order to enhance robustness, and, in particular, to strengthen the theoretical algorithm by adding consistency checks.

Our approach to implementing a robust algorithm is somewhat experimental. The first step is noticing what robustness difficulties arise. The second is analyzing their causes, and placing this analysis on a firm mathematical footing. In this paper, we concentrate on these first two issues, in an attempt to specify how such algebraic algorithms should be robust, and under what data and conditions. We point out causes of instability. Finally, where possible, we propose solutions for some of the cases in which we believe we have found a robust strengthening of the underlying theoretical algorithms.

We now describe the collision detection algorithms we implemented. These algorithms are essentially a specialization to $\Re^2 \times S^1$ of the six DOF collision avoidance algorithms of [Don 84, 87]. The basics of this geometric representation may be traced to [LoP, BLP]; in particular, see [BLP, Don 87] for a discussion of applicability constraints.

5.1 Representing Configuration Space Obstacles

We implemented a specialization of Donald's representation [Don 84, 87] for 6 DOF ($\Re^3 \times SO(3)$) configuration space obstacles to the case of $\Re^2 \times S^1$. We quickly review that representation. Let C denote the configuration space $\Re^2 \times S^1$, and let $(\boldsymbol{x}, \theta)$ be a typical configuration. Recall from [Don 87] that a Cspace obstacle CO is defined by a predicate on configurations

$$\bigwedge_{i \in cfamily(A,B)} \Big(\theta \in \mathcal{A}_i \Longrightarrow f_i(\boldsymbol{x}, \theta) \leq 0\Big). \tag{11}$$

Here, the functions $f_i : C \rightarrow \Re$ are called C-functions; *their negative conjunction* in effect defines the Cspace obstacle. Each f_i is restricted by an applicability region $\mathcal{A}_i$

which is a sector (angular interval $[\theta_{min}, \theta_{max}]$) of the unit circle S^1. The reason for this is that each f_i is generated by considering the interaction of a feature (edge or vertex) of a pawl polygon A with a feature (vertex or edge) of an obstacle polygon B. Contact, or interaction between generating features is only possible for a connected angular interval of orientations; this interval is precisely the applicability interval $\mathcal{A}_i$. The set of all C-functions generated by A and B is called the "C-family" $cfamily(A, B)$. See [BLP, Don87] for details on computing the C-functions and the applicability constraints.

A C-surface ker f_i is defined as the applicable zero-set (kernel) of a C-function f_i. It contains a patch of configurations where the two generating features of f_i can be placed in contact. The key step in detecting a collision of a path with a cspace obstacle CO defined by (11) is to simultaneously solve for the path's parameter subject to the csurface constraint[3] $f_i = 0$.

The special structure of the C-functions f_i permit us to define combinatorially precise algorithms for collision detection, and for solving general intersection problems. In particular (see, eg., [BLP] and apply [PD89]) the general form of a C-function in our application is

$$A_1xS + A_2xC + A_3yS + A_4yC + A_5S + A_6C + A_7x + A_8y + A_9 \tag{12}$$

where $\mathbf{x} = (x, y)$, $C = \cos\theta$ and $S = \sin\theta$.

Now, as is well-known (but see, eg., [Don 84]), we can make the substitution $u = \tan\frac{\theta}{2}$ in (12). Since $S = \frac{2u}{1+u^2}$ and $C = \frac{1-u^2}{1+u^2}$, we obtain a form of the csurface (12) which is quadratic in u:

$$B_1u^2 + B_2u + B_3 = 0 \tag{13}$$

where the coefficients B_i are all affine in x and y. When the root position $\mathbf{x}$ is parameterized by eq. (1), then note that the B_i are all affine in the time t, in the initial position ot the root $\mathbf{p}_0$, and in the root velocity vector, $\dot{\mathbf{p}}$.

Pure translational collision detection is straightforward; see, eg., [LoP, Don 87].

5.1.1 Computing the "Snap:" Pure Rotational Collision Detection

For pure rotational collision detection at a fixed translation $\mathbf{x}$, we implement the following algorithm. First, note that we must solve (13) for its u-zeros; these zeros determine the orientation at which the pawl can cross the boundary of the Cspace obstacle CO defined by (11). Each u-zero can be found by solving a quadratic (13). For each of these events, we construct an *intersection* which is a tuple $(\theta, f_i, \mathcal{A}_i, cfamily(A, B))$. These intersections are sorted around the unit circle (by θ). The intersections are traversed in order, and the first valid one is returned. A valid intersection is one where:

Intersection Validity Tests

[3]We will often use the term "constraint" to blur the distinction between a C-function and its kernel.

1. The constraint is zero: $f_i(x, \theta) = 0$. (This is true by root finding).
2. The constraint is applicable: $\theta \in \mathcal{A}_i$.
3. The intersection configuration is on the boundary of the CO (11). For all applicable f_i in $cfamily(A, B)$, $f_i(x, \theta) \leq 0$. Thus, we must test the signs of all other applicable constraints in f_i's family.

If A and B have size n and m, this algorithm runs in time $O(n^2m^2)$. To achieve the overall complexity bound of [PD89], we employ the $O(mn)$ algorithm of [Don 87]. We have given the slower algorithm here, because we have found that while it has higher asymptotic complexity it is more robust geometrically and stable numerically. This is probably because of the redundant information in the representation of a C-family.

Finally, since one C-family is generated for each convex-convex pawl-obstacle pair, this rotational intersection test must be iterated for each C-family, and the minimum valid intersection returned.

5.2 Collision Detection Subject to a Holonomic Constraint

A more complicated intersection problem arises when the pawl is sliding on a surface (13), subject to (1) (which specifies the motion of the root). Hence, we view the situation as follows. Suppose the pawl is sliding subject to a type B (vertex-edge) or type A (edge-vertex) constraint. This corresponds exactly to the reference point sliding on a quadric Cspace surface ker f_i which is generated by those contact features. The constraint on position (1) defines a "plane" in configuration space. The intersection of this plane and ker f_i form a quadratic curve in Cspace, parameterized by time. Using this parameterization, we can generate the configuration of the pawl at any time; that is, we have solved for how the pawl moves subject to the root motion (1) and subject to the Csurface (13). We must do this for each pawl (since, as in fig. 2, it is possible for both pawls to be in simultaneous contact with different features of the environment).

The first problem we must solve is how to detect when the quadratic path above intersects another csurface. At that point, the new constraint may take over, or the pawl may snap off due to incompatible kinematic constraints.

Each Csurface has form (13), where the coefficients B_i are affine in x, and hence affine in t, p_0, and $\dot{p}$ (see (1)). After [Don 84,87], we call this a *Trigonometric Quadratic Form (TQF)*. To solve for the simultaneous intersection of two Csurfaces in TQF, we view them as simultaneously quadratic in u and affine in t. We treat the variable t as indeterminate, and use the resultant to obtain a quartic in t. We solve for the t-roots, and back-substitute for the u-roots. Naturally, we must check for the degenerate case where the leading coefficients of the TQF's are zero (however, see sec. 5.3.1). Finally, given the (t, u)-roots, we must then perform the intersection validity tests described above in sec. 5.1.1 (or the faster test in [Don 87]). While there exist closed-form solutions to quartics, we solve them numerically using Ferrari's method. We view our implementation as a practically-motivated approximation to the theoretical, exact decision procedure.

5.2.1 Algebraic Procedures for Predicting Motion Subject to a Holonomic Constraint

In general, it is very hard to find exact, algebraic procedures for predicting motion subject to a holonomic constraint. This is because most dynamical systems do not lend themselves to such solutions. Our formulation of rotational compliance enjoys purely algebraic solution trajectories that can be computed using simple algorithms, and that require no integration. Of course, this is largely due to the simplicity of the dynamic model. Nevertheless, from the examples in sec. 3, it is clear that it is capable of producing complex behaviors.

5.3 The Genericity of Intersection Problems

Many algebraic robotics and motion planning algorithms make certain "genericity" or general position assumptions about the systems of polynomials they manipulate. We now discuss how certain of these assumptions are not necessary in our algorithm, because (in one sense), our constraints are "never singular:" (we make this notion precise below). On the other hand, there are other general position assumptions which would seem safe (and are, in fact, common in geometry) which, it turns out, are *not* valid in our application. We call this situation "forced non-genericity." Intuitively, it occurs with rotational compliance because of the tendency of the manipulated parts to rotate until many features are simultaneously aligned. In such cases, special techniques are needed to solve the non-generic intersection problems.

5.3.1 The Inherent Genericity of *C-functions* in Trigonometric Quadratic Form

In essence, our model permits a reduction of the motion prediction problem to solving polynomial systems of equations. Now, for arbitrary polynomials the leading coefficients of these systems can vanish. This singular situation is undesirable, since in these cases the resultant gives us no information, necessitating special checks. However, in the special case of TQFs that arise from C-functions, it turns out that we do not have to check explicitly for this case. More specifically, the degenerate case of the formal leading coefficients vanishing does occur. But if this happens, it means that in the case of TQFs, $\theta = \pi$ (mod 2π) is a solution for the trigonometric equations [Pai88]. This means that for Csurfaces in TQF, we can blindly apply elimination theory (and especially, the theory of resultants) to solve the polynomial systems. This fortuitous circumstance is only true in our special application, where the quadric surfaces come from TQFs like (13). Thus, while many algebraic algorithms require the system of polynomials to be in general position (this is called a genericity assumption), our method has no such genericity requirement.

Figure 6: The geometry of the flexible object and the environment are the same as in fig. 2. However, the assembly plan is to move diagonally in direction $(1, -\frac{4}{10})$ instead of straight down. The bottom vertex of the left pawl hits first. The next segment of the motion is subject to this constraint, in 6b. At the end of 6b, both pawl vertices lie on the edge. In 6c-d, the pawl is dragged over the top of the "T". In 6f-g, it snaps off to the rest position.

Figure 7: A detail of the bottom of the pawl from fig. 6.

5.3.2 On the Forced Non-Genericity of Intersection Problems Subject to a Holonomic Constraint

Whereas many algebraic motion planning algorithms can make assumptions about general position (or genericity) of the polynomial constraints, when we predict motions subject to a holonomic constraint, we find that these assumptions may not hold, and that we are forced to solve non-generic intersection problems.

See fig. 6. The geometry of the flexible object and the environment are the same as in fig. 2. However, the assembly plan is to move diagonally in direction $(1, -\frac{4}{10})$ instead of straight down. See fig. 7 for a detail of the bottom of the pawl. The bottom vertex v_i of the left pawl hits the obstacle edge e first. The next segment of the motion is subject to this constraint. That is, the motion is compliant subject to the csurface ker f_i generated by (v_i, e). Now, refer fig 7 again, and consider an *arbitrary* (generic) motion of the triangle (a path in Cspace). Obviously, this path can cause either constraint (v_i, e) or (v_j, e) to be violated. However, we say that *generically*, both will not be violated at the same time. That is, for a path $\phi : [0, 1] \rightarrow C$, it will be generically true that $f_i(\phi(t))$ or $f_j(\phi(t))$ is non-zero. (f_j is generated by (v_j, e)). This would be a good general position assumption, but unfortunately, it is simply not true for collision detection subject to a rotationally compliant motion constraint. In fig. 6, we see that as the motion evolves, A rotates about v_i as v_i slides on e. Eventually, this rotation brings v_j down on e. In fact, constraint f_i expires (that is, we pass out of its applicability region) at the precise time that f_j is activated (we pass into its applicability region); at the same time, f_j changes sign from positive to zero.[4] More precisely, at this time t, the orientation $\theta(t)$ crosses the boundary of $\mathcal{A}_i$ and $\mathcal{A}_j$ (which share an endpoint), and $\phi(t)$ hits the zero set of f_j. See fig. 8.

The non-genericity illustrated in figs. 6-8 in fact occurs for a large (generic) class of paths and initial conditions. This is not surprising, since of course one of the outcomes of rotational compliance is to align parts. Second, it is not surprising that when one constraint (C-function) expires, it is "replaced" by a constraint with "neighboring generators." This observation has been formalized by [Don 84, 87] and also exploited by

[4]We use "expire" and "become active" following [Don 87].

Figure 8: ker f_i is the csurface for contact (v_i, e) (see fig. 7). ker f_j is the csurface for contact (v_j, e). The rotationally compliant path ϕ is following ker f_i, when it simultaneously (i) strikes the boundary of f_i's applicability region $\mathcal{A}_i$, (ii) pierces the boundary of f_j's applicability region $\mathcal{A}_j$, and (iii) hits the zero-set of f_j. ϕ is shown schematically in solid lines.

[ELP]. For these reasons, naturally, it would seem as if non-generic examples such as fig. 6 would happen all the time with our system. In fact, this has been our experience. It is unfortunate, in that it places high demands on the robustness of our algebra system. In particular, the applicability constraint boundaries of $\mathcal{A}_i$ and $\mathcal{A}_j$ are, in fact, computed as the zeros of polynomials also (see [Don 84, 87]). Hence, in observing that three events occur simultaneously (fig. 8) we are effectively saying that three polynomials (in t) have simultaneous zeroes. When these zeroes appear to occur at different times because of numerical errors, then consistency is not maintained and errors can occur. For example, if we find that ϕ crosses ker f_j before it crosses the applicability boundary $\mathcal{A}_i \cap \mathcal{A}_j$, then that intersection will be judged non-applicable and discarded. If the events are ordered the other way, the intersection is detected.

We can attempt to maintain consistency by introducing additional tests into the algorithm, based on topological information. For example, (see [Don 87]), the polynomial g_{ij} whose zeros define the applicability boundary $\mathcal{A}_i \cap \mathcal{A}_j$ is essentially $f_i - f_j$. Hence, at any given configuration, there are consistency constraints between the signs (and values) of f_i, f_j, and $g_{ij} = f_i - f_j$. Second, we could employ the techniques of [Don87, ELP], which compute, when a constraint "expires", which constraints with neighboring generators become applicable. Thus, when f_i expires at time t, we have $\phi(t) = \mathcal{A}_i \cap \mathcal{A}_j$. We can look at $\dot{\phi}(t)$ and the neighboring generators of v_i and e, to determine that f_j must "replace" f_i in the applicability set (see fig. 7).[5] Finally, we must perform these constraint replacement computations "subject to a holonomic constraint". That is, if f_j replaces f_i in free space, it suffices to simply substitute f_j for f_i in the applicability set. However, in contact, subject to the holonomic constraint

$$f_i(x, \theta) = 0 \tag{14}$$

we must not only update the applicability set, but we must also update the constraint (14). That is, we must simultaneously replace f_i by f_j in the applicability set, and replace (14) by the constraint $f_j = 0$. This constraint replacement mandates that $f_j = 0$. (This is equivalent to the saying that we maintain contact). Hence, we know that at time t, we should simultaneously have f_i, f_j, and g_{ij} vanishing. If because of numerical errors these zeroes occur at different times, then we should in effect "identify" these times into one canonical, simultaneous zero-crossing time for topological consistency.

Finally, we note that the issue of forced non-genericity is subtle mathematically, and has connections to other branches of singularity theory. For example, consider fig. 9. In

[5]The *applicability set* is the set of all constraints (C-functions) that are applicable at a configuration.

Figure 9: Abstract depiction of forced non-genericity. In a configuration space C consider a lower dimensional algebraic singular set S of "bad" points that we wish to avoid. For example, S could be the set of singular configurations of the forward kinematic map.

Figure 10: There are two orientations (u-roots) at which the C-function f_i generated by (v_i, e) is zero. Which of these two solutions should we pick?

general, we have a configuration space C containing a lower dimensional singular set S that is algebraic. (In our case, S is the boundaries in fig. 7). Now, if we choose two points z_1 and z_2 randomly in C, the chance that one will lie in S is essentially zero (this is really the definition of measure zero). However, if we now fix z_1 and z_2 and consider all paths from one point to the other, it turns out that "many" of them—a measurable set—cross S. In particular, if S separates C, then one might expect this probability to be roughly proportional to the ratio of the measure of the two components of $C - S$. Finally,

Fact: *If z_1 and z_2 are disconnected by S in C, then any path from z_1 to z_2 will cross S.*

This means that any such path must experience a singular configuration at some time. This situation is exactly parallel to ours, in which a singularity cannot be avoided. Note that this problem arises in other applications—for example, in [Pai88], we observed S as the singular set of a forward kinematic map. By the claim above, this means (for example) that any path from z_1 to z_2 must pass through a singularity.

We have sketched an approach to handling forced non-genericity that exploits the specific characteristics of our domain. It may be that extensions of [Pai88] may provide general techniques for handling these problems. See [Pai88] for a general discussion and more observations on this phenomenon.

5.4 Choosing the Correct Branch

This section deals with the following difficulties. When we reduce motion prediction problems to the existential theory of the real numbers, we obtain our answers by solving polynomial equations. These equations may have multiple roots. However, our system only has one evolution. We must have a method for choosing which root is correct. Conversely, situations will arise where the algebraic equations fail to have roots. We must handle these cases also.

The situation of multiple roots corresponds precisely to the existence of several "statically" or "kinematically feasible" solutions to the dynamical system, only one of which is feasible (or reachable) from the initial conditions. We present a method for choosing the correct root. When the correct root is parameterized by time, it sweeps out a connected component called a "branch." This occurs in our case, where the formal coefficients are parameterized by t.

The absence of roots corresponds precisely to the pawl breaking contact with a Csurface, or jamming on the Csurface so that no further motion is possible. We show how to determine which of these events has occurred. These cases are important to handle in practice, and our methods are both theoretically well-founded and implementable.

Consider fig. 10. Clearly, when we solve a TQF such as (13) for its u-roots, we are solving a quadratic; hence we obtain two roots. Now, often one root will be inapplicable, or not on the boundary of the cspace obstacle CO (eq. (11)). However, in fig. 10, both are good roots. Which one should we pick? This is an important question for an implementation. First, we note that even though both are good roots, there is no realizable path between them (and so they are disconnected). Consider the problem of generating a path in cspace subject to csurface constraint (13) and to the straight-line motion (1). (This corresponds to intersecting a quadric csurface with a plane to obtain a quadratic curve, parameterized by time). The path is constructed using elimination methods; consider the problem of "following" this path. One reason we need a systematic way of choosing the correct root is that, due to numerical errors, often one root will appear to become inapplicable "too early" (see, for example, sec 5.3.2), or else, a root appears to just slide off the boundary of the cspace obstacle. At this point the implementation finds the other root (as in fig. 10). This root is applicable, and on the boundary, and so the predicted path "jumps" discontinuously from one root to the other, even though no such path is physically feasible.

This example illustrates pathological behavior. Fortunately, we can give a principled way for choosing the correct root. Essentially, one is finding zeros of a simple algebraic variety in the plane; the problem is equivalent to choosing the correct "branch" of the variety. We show how to do this algorithmically. In the process, we also derive a method for determining when a pawl "jams" against a constraint, and when a pawl "snaps off" a constraint due to incompatible differential kinematics. The two prove to be related problems.

Let us assume that the Csurface is in the form (from (13))

$$S(u,t) = A(t)u^2 + B(t)u + C(t) = 0 \tag{15}$$

where

$$\begin{aligned} A(t) &= A_0 + A_1 t \\ B(t) &= B_0 + B_1 t \\ C(t) &= C_0 + C_1 t \end{aligned} \tag{16}$$

and $u = \tan\frac{\theta}{2}$ as usual. Then the branch problem is essentially the problem of choosing the correct sign of the radical in

$$u = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \tag{17}$$

Figure 11: Plotting the discriminant $\Delta(t) = B^2 - 4AC$ vs. time, for type (B) and (A) constraints. We illustrate the qualitative regions in the space. The "pawl" here is represented as follows. For a type (B) constraint, the pawl is a straight line connecting the pivot p with the contact vertex. The obstacle edge is stationary. For a type (A) constraint, the edge moves rigidly with p, and v is a stationary obstacle vertex.

Figure 12: If t_{z_n} is finite, we must now check the behavior at this time. Two things can happen as t is increased a small amount from t_{z_n}. We can break contact (i) or jam (ii). See fig. 11. p is the pivot point, and $\dot{p}$ the pivot velocity. Type (B) and (A) contacts are shown. Case (i-B) (break contact) shows the friction cone. Case (i-A) is impossible, and cannot occur unless we started from this configuration.

The problem of determining when the pawl snaps off or jams is essentially that of determining if the discriminant $\Delta(t) = B^2 - 4AC$ is becoming negative, and hence failing to have a real solution. Let us discuss this second case first, and then we will return to discuss the branch problem (how to choose the sign of $\sqrt{B^2 - 4AC}$) afterwards.

5.4.1 Application: Snapping Off and Jamming

See fig. 11, plotting the discriminant $\Delta(t) = B^2 - 4AC$ vs. time, for type (B) and (A) constraints. We illustrate the qualitative contact regions in the space. In the type (B) case, the "pawl" here is drawn as a straight line segment, pivoting about its end point. (The pivot is the center of rotation of the pawl—where it would be attached to a root body). The other endpoint is in contact with an edge. A type (B) constraint is a csurface generated by a vertex of the moving pawl and an edge of the obstacle. A type (A) constraint is generated by an edge of the moving pawl and a vertex of the obstacle. See [PD89]. Hence in the type (A) figure, the edge moves with the pivot p and the vertex v is stationary, whereas in the type (B) figure, the edge is stationary, and the vertex at the end of the line segment pawl moves.

Now, the discriminant $\Delta(t)$ is non-negative if and only if eq. (15) has a real solution. hence we need the following test for the zero crossings of the discriminant $\Delta(t)$, which is obtained by solving the quadratic $\Delta(t) = 0$ for its root t_z. That is,

$$\begin{aligned} \Delta(t) &= B^2 - 4AC \\ &= (B_1^2 - 4A_1C_1)t^2 + 2(B_0B_1 - 2A_0C_1 - 2A_1C_0)t + B_0^2 - 4A_0C_0. \end{aligned} \tag{18}$$

This will produce two times t_{z_1} and t_{z_2}, with $t_{z_1} \leq t_{z_2}$. Clearly, the time of the first collision with this surface, t_c, must satisfy $t_{z_1} \leq t_c \leq t_{z_2}$ for type (B) csurfaces, and $t_{z_1} \geq t_c$ or $t_c \geq t_{z_2}$ for type (A). Therefore, the *next* critical time t_{z_n} is $t_{z_n} = t_{z_2}$ for type (B) csurfaces. For type (A) csurfaces, $t_{z_n} = t_{z_1}$ if $t_c \leq t_{z_1}$. Otherwise, $t_{z_n} = \infty$ if $t_c \geq t_{z_2}$ (i.e., there is no critical time).

If t_{z_n} is finite, we must now check the behavior at this time. Two things can happen as t is increased a small amount from t_{z_n}. See fig. 12. The pawl-obstacle contact is

Figure 13: Non-generic case of translation parallel to the edge.

represented as in fig. 11. For either type (B) or (A) contact, we can either (i) break contact or (ii) jam. The figure shows these possibilities.

Now, which behavior occurs can be determined from the Csurface equation. We assume that the sign of the equation is the same as in [Don87,PD89], i.e., $S(u,t) > 0$ means that the vertex is outside the half-space bounded by the edge.

Equation (15) for $S(u,t)$ can be rewritten as

$$S(u,t) = (A_1u^2 + B_1u + C_1)t + (A_0u^2 + B_0u + C_0) \tag{19}$$

First, we solve for u_z at time t_{z_n} from eq. (15). There should be only one solution:

$$u_z = \frac{-B(t_{z_n})}{2A(t_{z_n})} = \frac{-(B_0 + B_1t_{z_n})}{2(A_0 + A_1t_{z_n})}. \tag{20}$$

Next, we examine the sign of the coefficient of t in eq. (15) for this value of u_z. Let $\sigma = A_1u_z^2 + B_1u_z + C_1$.

1. If $\sigma \geq 0$, then the pawl breaks free. We perform a "snap" (pure rotation) towards the zero position.
2. Else, $\sigma < 0$. The pawl is jammed. Report this and terminate the prediction.
3. Note: $\sigma = 0$ is the non-generic case of translating parallel to the edge (see fig. 13). This should probably be considered a "snap", but the exact semantics are somewhat unclear.

5.4.2 The Branch Problem

We now return to the branch problem. This is essentially the problem of choosing the correct sign of the radical in eq. (17). This may be done as follows. First we define the following algorithm, which computes the branch sign for a Csurface at time t and orientation u.

```
Algorithm Branch-Sign(u, t, Csurface)
1. Let $A(t)$, $B(t)$, $C(t)$ be the coefficients of the Csurface.
2. $x \leftarrow -\frac{B(t)}{2A(t)}$.
3. $y \leftarrow \frac{1}{2A(t)}\sqrt{B^2(t) - 4A(t)C(t)}$.
```

Figure 14: With uncertainty, the root position starts out in some region R. The control system ensures that as time evolves, the root position stays in a cylinder $T_{\dot{p}}$.

4. If $u = x + y$ *then return* Plus.[6]

5. If $u = x - y$ *then return* Minus.

Now, every time we move onto a new Csurface, we record its sign *Branch-Sign*(u_c, t_c, *Csurface1*). When new candidate intersections are produced when sliding along *csurface1*, then we check that the corresponding u_{new} and t_{new} for the intersection fall on the branch of *csurface1* with the same branch-sign. That is, we check whether or not *Branch-Sign*(u_c, t_c, *Csurface1*) = *Branch-Sign*(u_{new}, t_{new}, *Csurface1*).

6 Incorporating Uncertainty in Control and Initial Conditions

6.1 Assumptions

Real robots are subject to significant uncertainty and error in sensing and control. For this reason we would like to generalize our algorithm to deal with uncertainty in initial conditions, and uncertainty in control. In this section we show how this may be done, using a simple model of uncertainty. The introduction of uncertainty changes the complexity of our algorithm from $O(N^2 \log N)$ to $O(N^3 \log N)$; the algorithm remains combinatorially precise and algebraic.

First, we assume that the initial position p_o of the root body is known to start within some "start region" R. This region represents uncertainty in the initial conditions. We will still assume that the root body is controlled as a "moving constraint", parameterized by time. However, we assume that it is controlled by something like a generalized spring controller with feedback position correction. This model is very similar to that analyzed by Buckley (the spring-damper model) [Buc87], and we adopt it because Buckley's error analysis is convenient algorithmically. However, we will not use the damper component of the model to slide on surfaces. Under these assumptions, the set of possible positions that the root position p can reach (in free space, without obstacles) is bounded by a cylinder. In short, we incorporate sensing in control by assuming that with initial conditions p_o in R, the set of positions that the root position p can reach is contained in a cylinder $T_{\dot{p}}$ starting at R. See fig. 14.

See eq. (1). The geometric formalization of our assumptions is that (i) $p_o \in R$, and (ii) for all times t, the root position $p(t)$ lies in a cylinder $T_{\dot{p}}$ from R as in fig. 14. The

[6]With finite precision arithmetic, these equality checks cannot be exact.

practical realization of these assumptions requires (i) a bound on sensing errors so that the initial position can be ascertained to some known accuracy, and (ii) a feedback control system with bounded error, such as that of [Buc87].

6.2 Algorithms

6.2.1 Motion Prediction under Uncertainty is Computable

It is immediately clear that with this formulation of uncertainty, the motion prediction problem (to predict all possible outcomes from R subject to $T_{\dot{\mathbf{p}}}$) is decidable. We see this as follows. First, our algorithm for motion prediction from a single initial condition given $\dot{p}$ (as described in [PD89] and above) defines an algebraic predicate $F_{\dot{\mathbf{p}}}(p_o, \mathbf{z})$ that decides whether a configuration $\mathbf{z}$ is reachable from initial condition p_o given commanded motion $\dot{p}$. In effect, our algorithm computes the entire set

$$\exists \mathbf{z} \quad F_{\dot{\mathbf{p}}}(p_o, \mathbf{z}). \tag{21}$$

That is, our algorithm outputs the entire set $Z(p_o, \dot{p})$ satisfying (21). Thus we can define another predicate

$$\mathcal{F}(x, \mathbf{z}) \iff (\exists \mathbf{z})\,(\exists x \in R) \quad F_{\dot{\mathbf{p}}}(x, \mathbf{z}). \tag{22}$$

Correspondingly, predicates can be defined to decide all surfaces on which the pawls might stick, all surfaces the pawls might slide on, etc. Since the predicates are algebraic, they are decidable.

6.2.2 A Practical Alorithm

More practically, we can define a direct algorithm with a computational geometric flavor. Assume that generically, only one initial contact can occur at a time. Imagine that we "project" R onto the obstacle environment in "direction" $\dot{p}$. This is similar to "intersecting" $T_{\dot{\mathbf{p}}}$ with the environment. That is, we sweep the flexible body along $\dot{p}$ from R, and find all edges of the environment it can hit. We find all initial contact tuples

$$(g_A, t_{c_{min}}, t_{c_{max}}, g_B, x(t_c))$$

such that pawl feature g_A can strike obstacle feature g_B at times t_c, $t_{c_{min}} \leq t_c \leq t_{c_{max}}$. Note that at initial contact of (g_A, g_B), the orientation of g_A and the other pawls is unchanged. $x(t_c)$ is a function that maps initial contact times to root positions. Hence, the initial contact tuple represents the fact that for $t_{c_{min}} \leq t_c \leq t_{c_{max}}$, pawl feature g_A can strike obstacle feature g_B if the root position is at $x(t_c)$.

For example, in fig. 14, assume the pawl is a line segment anchored at p (as in fig. 11). There would be three initial contact tuples, one for each type (B) csurface of $\{v\} \times \{1,2,3\}$. We call the set of initial contact tuples the *projection* $\pi_{\dot{\mathbf{p}}}(R)$ of R under $T_{\dot{\mathbf{p}}}$.

Given the initial contact tuples, we chose a "sample point" for each one, and run our standard, no-uncertainty algorithm from that sample point. The algorithm is given below.

```
Algorithm for Motion Prediction Under Uncertainty

1. Compute the initial contact tuples $\pi_{\dot{p}}(R)$.
2. For each initial motion tuple $(g_A, t_{c_{min}}, t_{c_{max}}, g_B, x(t_c))$, do
3.   Assume type (B) contact, so that $g_B$ is an obstacle edge.[7] Segment $g_B$ into 3 regions:
4.       The sliding region $e_1$,
5.       The sticking region $e^*$
6.       and the portion $e_0$ of $e$ that cannot be hit initially.
7.   Output $e^*$.
8.   Choose a sample point $x \in e_1$.
9.   Compute and output the reachable set $Z(x, \dot{p})$ (eq. (21)) using our motion prediction algorithm for perfect initial conditions and no uncertainty.
```

Let N be the geometric complexity, that is, the product of the moving object features and obstacle features. Then there are at most $O(N)$ initial contact tuples. Thus the loop is executed $O(N)$ times. e_1, e^*, and e_0 have size at most 2. Hence there are at most $O(N)$ sample points x, and so the basic algorithm is called $O(N)$ times as a subroutine in the last step. Since the basic algorithm runs in time $O(N^2 \log N)$, that means the algorithm for motion prediction under uncertainty takes time $O(N^3 \log N)$.

6.3 Extensions and Limitations

Ideally, one would like to extend our model of uncertainty to be more realistic. Ideally it should cover uncertainty in the direction of the moving constraint, and uncertainty in initial orientation. As of now, we hope it is rich enough to model interesting phenomena, but still be computationally feasible. Much work remains to be done in reasoning about assemblies in the presence of significant uncertainty. Our initial algorithm is just a start.

[7]The case for type (A) contact is very similar.

7 Conclusions

In this paper we described a system for reasoning about and analyzing the motion of a "flexible object"—i.e., a compliantly-connected system of rigid bodies. We reviewed the basic theory developed in [PD89] on the motion prediction problem for such bodies, at a fairly abstract level, emphasizing connections to computational mechanics and the long-term behavior of dynamical systems. We discussed how our formulation of the problem led to theoretically precise algorithms for motion prediction with rotational compliance, and showed how these algorithms could be implemented and used to analyze designs for assembly. We stressed the fact that while the theoretical algebraic algorithms we give are correct when exact arithmetic is employed, in practice we must strengthen the algorithms to make them robust when implemented using finite-precision arithmetic. We showed how this may be done in a systematic fashion. Issues of robustness and numerical stability turned out to be related to problems of genericity and branch-choice, and we described the theory behind these problems and their practical solution. This elaboration of the algorithm sketched in [PD89] completes the description of the computational approach, with an emphasis on the pitfalls and subtleties that an implementation foregrounds.

The research contributions of this paper include:

1. New algebraic techniques for predicting the motion of objects in contact under our model of rotational compliance.

2. A systematic catalog of singular and non-generic situations that must be handled, and algorithms for dealing with these events.

3. An extension of our algorithm to compute mating force information. This facility can be used to design objects that are easier to mate than to disassemble.

4. An extension of our algorithm to efficiently predict motions given uncertainty in sensing and control.

5. A manifesto for the relevance of our approach to engineering, and particularly to design for assembly. As an application, we studied several classes of flexible devices that we term "motion diodes." We developed a classification of these devices into total vs. relative motion "diodes", and described kinematic, frictional, and force diodes. Our algorithm can analyze designs, and classify the diode type. We showed how motion diodes could be useful in design for assembly.

Much remains to be done. We wish to extend and test our system in analyzing real designs. We also intend to fabricate parts we have designed, and test them by assembling them with real robots. Perhaps the most challenging area for the future involves the interaction of our analysis approach with uncertainty. We hope to extend our analysis and algorithms of sec. 6 to model uncertainty more precisely and to generate designs that can be assembled robustly. In particular, we hope to develop a precise notion of the "stability" of a motion with respect to a design. For example, an assembly plan might be "stable" if the qualitative outcomes of execution are equivalent. Another direction for future research would involve extending our analysis to articulated bodies (our flexible objects are trees of depth one; however, one could imagine many compliantly-connected links in a chain). This would be of practical interest. Finally, we hope to add more sophisticated dynamics to our work, and to model continuously deformable bodies as well.

We feel that algorithmic research on design for assembly represents a new direction in robotics which could significantly impact the field. In the future, one could imagine our techniques employed in an approach to *" design as search"*, in which we modify and improve an existing design by changing its geometry incrementally and applying our analysis algorithm. For example, suppose we are given geometric models of two parts to mate, and an approximate "assembly plan" (direction of mating). We envision an algorithm that searches around the boundaries of the models, and modifies the shape by introducing snap-fasteners and ratchet and pawl mechanisms. After each modification, the analysis algorithm described above could be run, to determine how the parts will mate, and the forces required to mate and disassemble them. Depending on the results of the analysis, the design change could be modified, retracted, or declared suitable. Of course, there is a host of conceptual and algorithmic problems that must be solved to make such an approach possible, but we believe that with the algorithmic underpinnings described in our analysis algorithm, a research program in this area is now viable. Such research could help to put design for assembly on a firm algorithmic footing, build software tools for generating good designs, and result in a unified framework for designing and assembling mechanisms.

Acknowledgments: We would like to thank Randy Brost for very helpful comments on these ideas and on a draft of [PD89], and to thank Mike Caine for helpful discussions.

8 References

Briggs, A. *An Efficient Algorithm for One-Step Compliant Motion Planning with Uncertainty,* 5th ACM Symposium on Computational Geometry, Saarbrucken, Germany. (1989).

[BLP] **Brooks, R., and T. Lozano-Pérez,** "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", *Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, Germany, August, 1983.

[BM] **Brost, R. and Mason, M.** *Graphical Analysis of Planar RIgid-Body Dynamics with Multiple Frictional Contacts,* Preprints, 5th Int'l. Symp. on Robotics Research, Tokyo, Japan, August. pp 367-374 (1989).

[Buc87] **Buckley, S. J.** *Planning and Teaching Compliant Motion Strategies,* Ph.D. Thesis. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1987. Also MIT-AI-TR-936 (1987).

[C] **Canny, J. F.** *Collision Detection for Moving Polyhedra,* PAMI-8(2) (1986).

[Can89a] **John Canny.** *On computability of fine motion plans.* , IEEE ICRA, Scottsdale, AZ, (1989)

[Don84] **Donald, B. R.** *Motion Planning with Six Degrees of Freedom,* MIT AI-TR 791, Artificial Intelligence Lab. (1984).

[Don85] **Donald, B. R.** *On Motion Planning with Six Degrees of Freedom: Solving the Intersection Problems in Configuration Space,* IEEE International Conference on Robotics and Automation, St. Louis, MO (1985).

[Don87] **Donald, B. R.** *A Search Algorithm for Motion Planning with Six Degrees of Freedom,* Artificial Intelligence, 31 (3) (1987).

[Don88a] **Donald, B. R.** *The Complexity of Planar Compliant Motion Planning with Uncertainty,* Proc. ACM Symposium on Computational Geometry, Urbana, IL June, 1988. *To appear in* Algorithmica. (1988).

[Don88b] **Donald, B. R.** *A Geometric Approach to Error Detection and Recovery for Robot Motion Planning with Uncertainty,* Artificial Intelligence 37 (1-3), Dec. 1988. pp. 223-271. (1988).

[Erd84] **Erdmann, M.** *On Motion Planning With Uncertainty,* MIT AI Lab, MIT-AI-TR 810 (1984).

[Erd86] **Erdmann, M.** *Using Backprojections for Fine Motion Planning with Uncertainty,* IJRR Vol. 5 no. 1 (1986).

[ELP] **Erdmann, M. and Lozano-Pérez, T.** *On Multiple Moving Objects,* Algorithmica, (2): 477-521 (1987).

[LoP] **Lozano–Pérez, T.** *Spatial Planning: A Configuration Space Approach,* IEEE Trans. on Computers (C–32):108–120 (1983a).

[LMT] **Lozano-Pérez, T., Mason, M. T., and Taylor, R. H.** *Automatic Synthesis of Fine-Motion Strategies for Robots,* Int. J. of Robotics Research, Vol 3, no. 1 (1984).

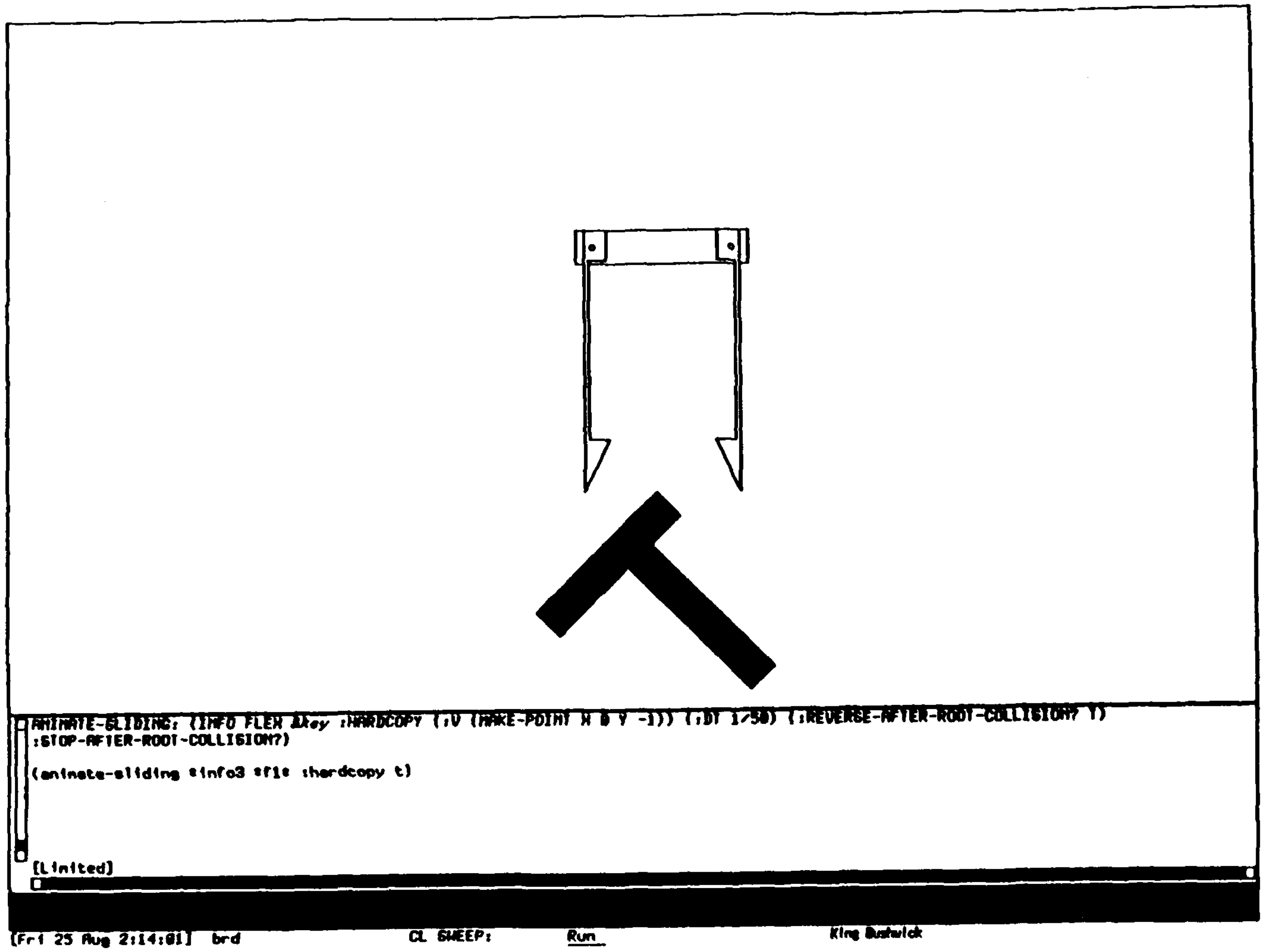
[Ma84] **Mason, M.** *Personal Communication,* (1984).

[Mas86] **Mason, M. T.** 1986. Mechanics and Planning of Manipulator Pushing Operations. *International Journal of Robotics Research* 5(3).

[Pai88] **Dinesh K. Pai.** Singularity, Uncertainty and Compliance of Robot Manipulators. , PhD thesis, Cornell University, Ithaca, NY, May 1988.

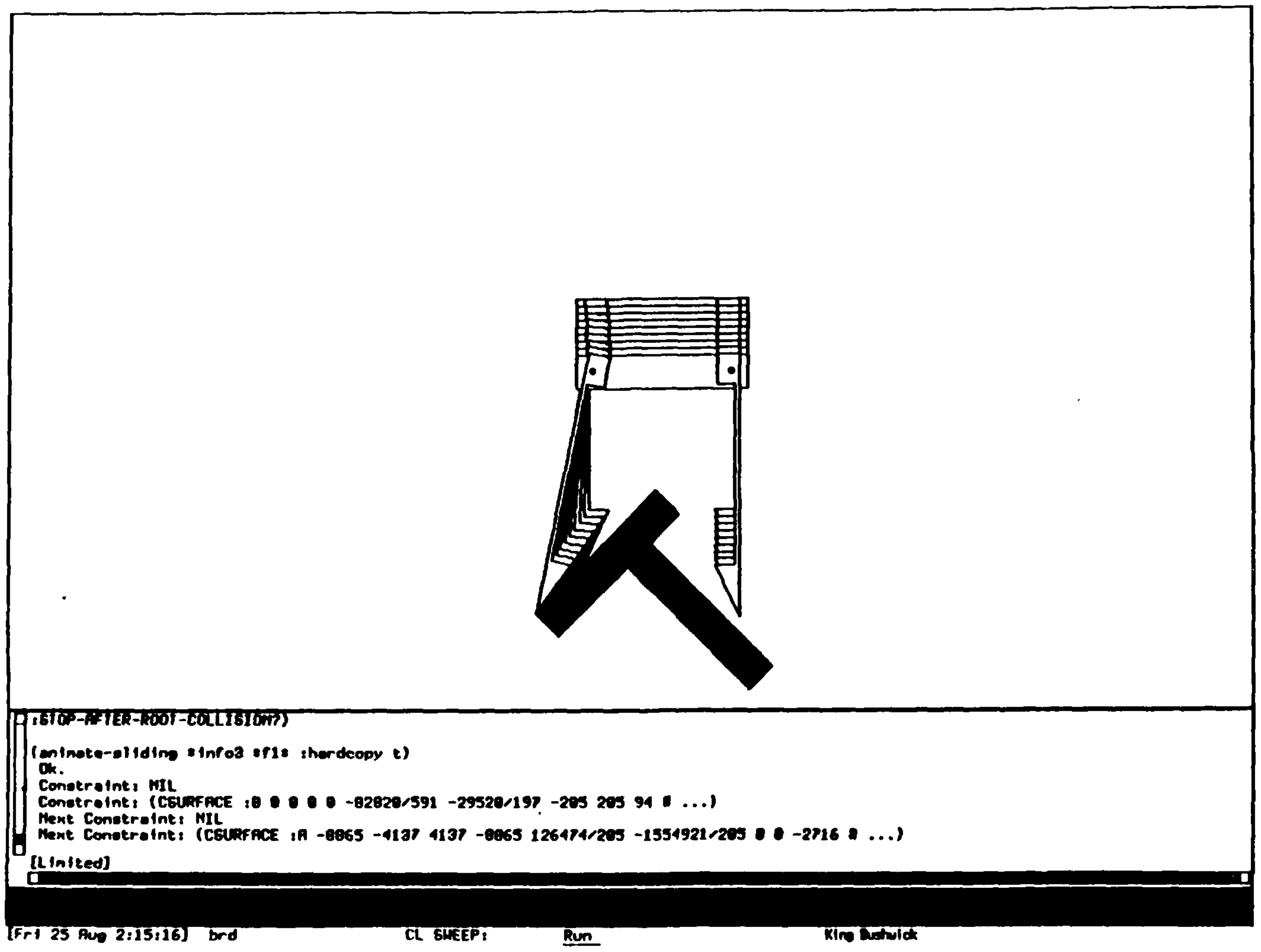
[PD89] **Pai, D. and B. Donald** *On The Motion of Compliantly-Connected Rigid Bodies in Contact, Part I: The Motion Prediction Problem,* Cornell Computer Science Technical Report, Submitted to IEEE Int'l. Conf. on Robotics and Automation, 1989 (1989).

[Whi82] **Whitney, D.**, "Quasi-Static Assembly of Compliantly Supported Rigid Parts", *Journal of Dynamic Systems, Measurement, and Control,* Vol. 104, March 1982.



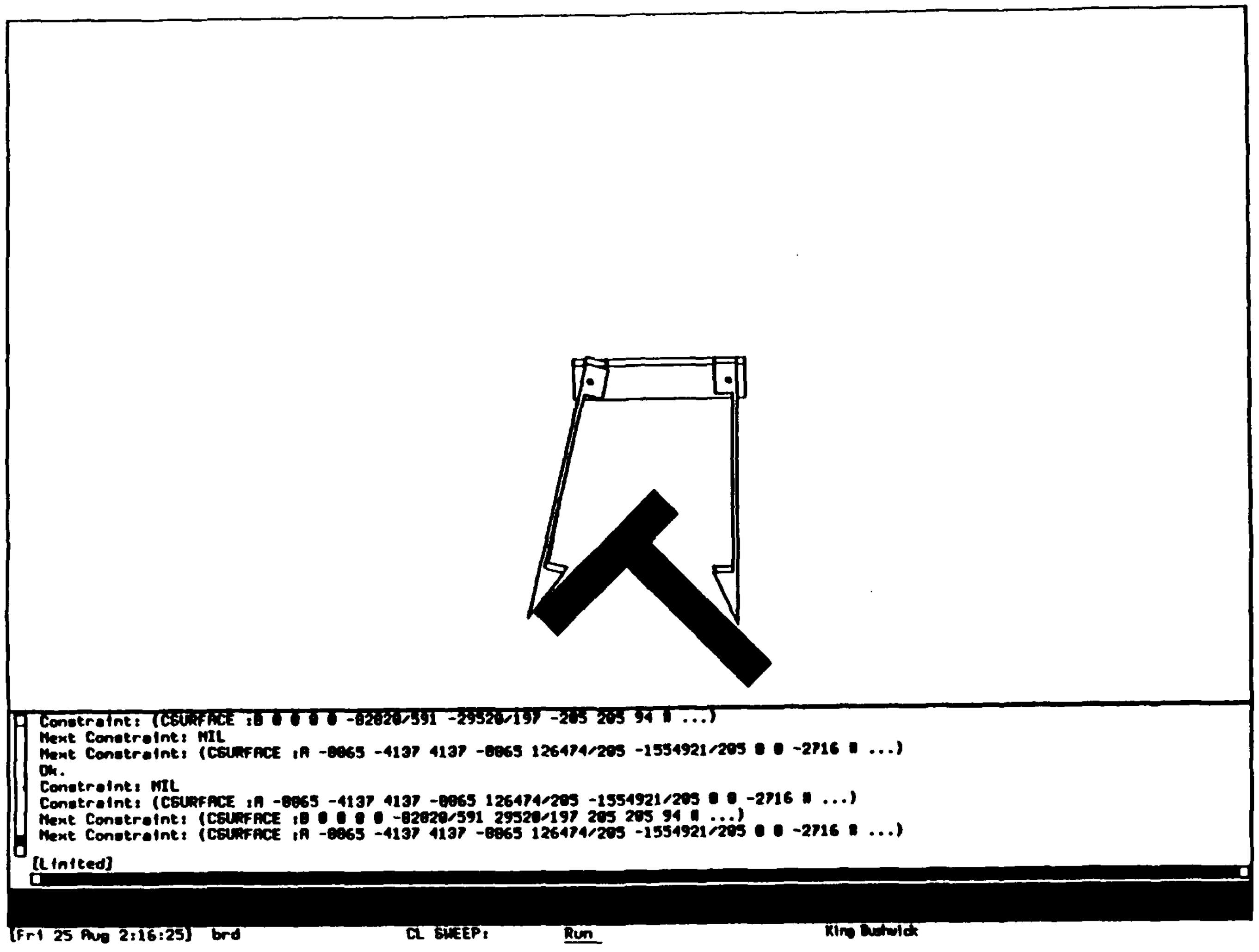
```
ANIMATE-SLIDING: (INFO FLEX &key :HARDCOPY (:V (MAKE-POINT X 0 Y -1)) (:DT 1/50) (:REVERSE-AFTER-ROOT-COLLISION? T)
:STOP-AFTER-ROOT-COLLISION?)

(animate-sliding *info3 *f1* :hardcopy t)



[Limited]

[Fri 25 Aug 2:14:01]   brd            CL SWEEP:     Run              King Bushwick
```

fig 1



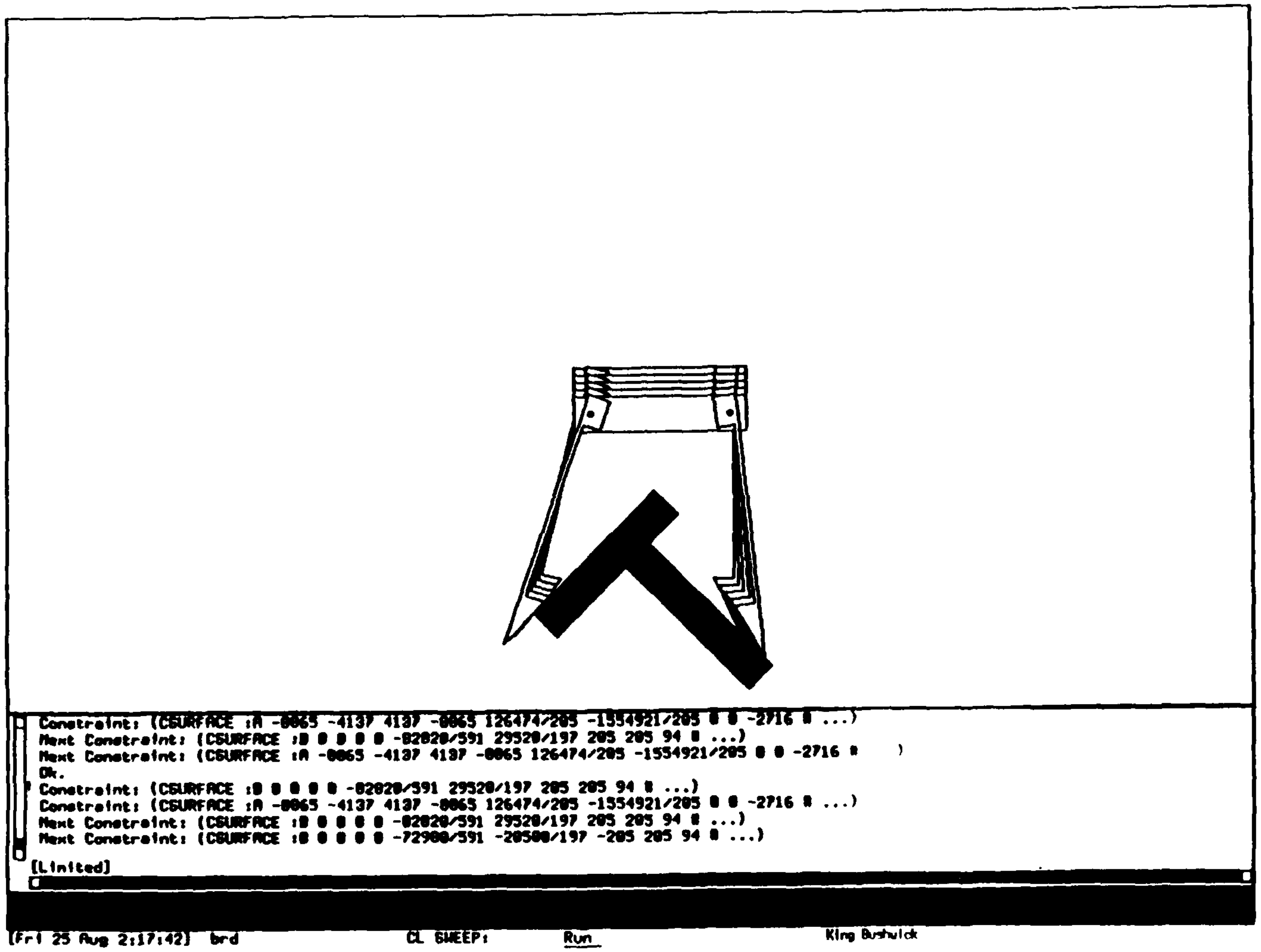
```
:STOP-AFTER-ROOT-COLLISION?)

(animate-sliding *info3 *f1* :hardcopy t)
 Ok.
 Constraint: NIL
 Constraint: (CSURFACE :0 0 0 0 0 -82820/591 -29520/197 -205 205 94 0 ...)
 Next Constraint: NIL
 Next Constraint: (CSURFACE :A -8865 -4137 4137 -8865 126474/205 -1554921/205 0 0 -2716 0 ...)
[Limited]
```

fig 2a



```
Constraint: (CSURFACE :B 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Next Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 0 ...)
[Limited]
```

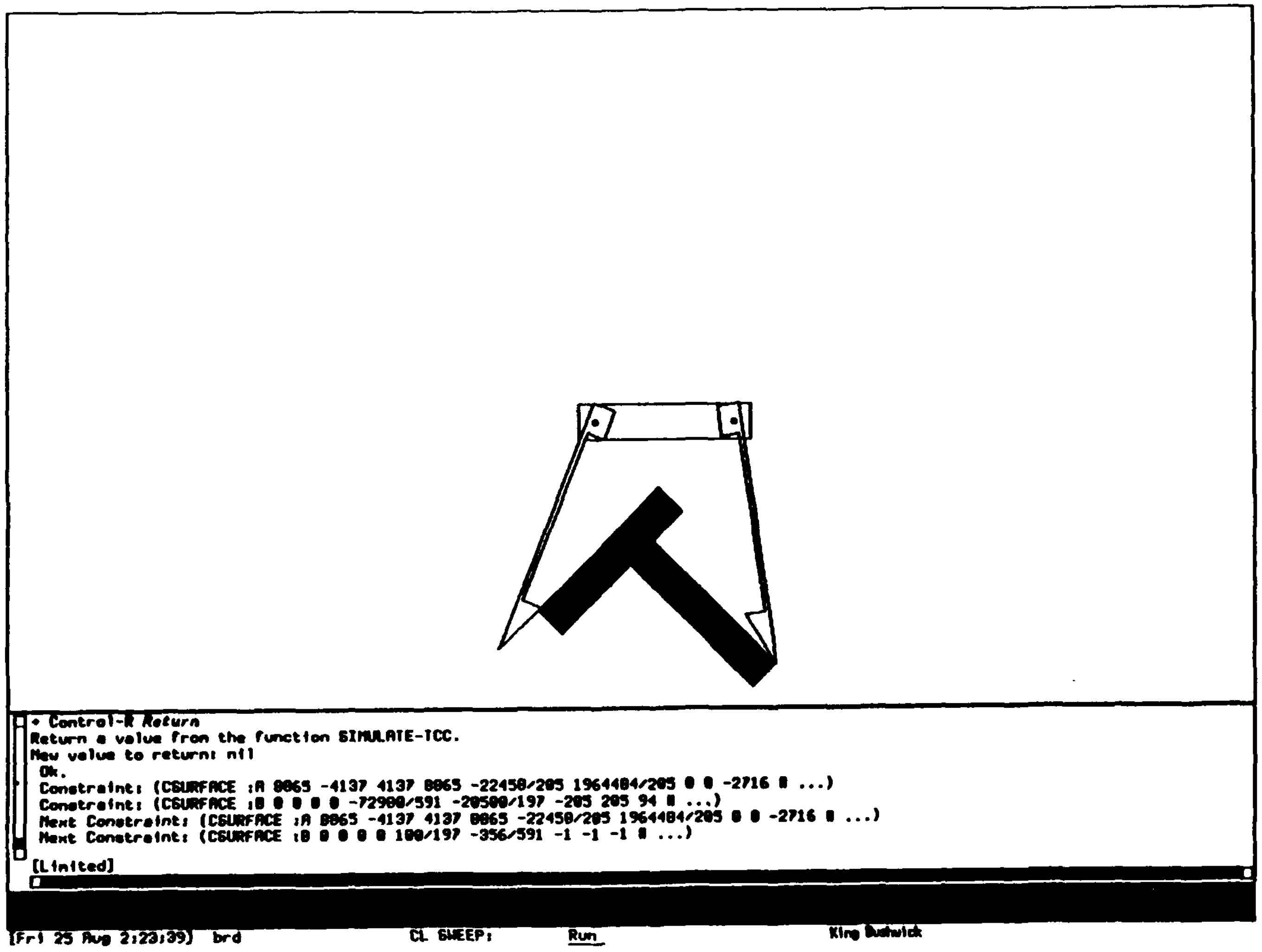
Fig 2b



```
Constraint: (CSURFACE :A -0065 -4137 4137 -0065 126474/205 -1554921/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Next Constraint: (CSURFACE :A -0065 -4137 4137 -0065 126474/205 -1554921/205 0 0 -2716 0     )
Ok.
Constraint: (CSURFACE :0 0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Constraint: (CSURFACE :A -0065 -4137 4137 -0065 126474/205 -1554921/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)

[Limited]
```

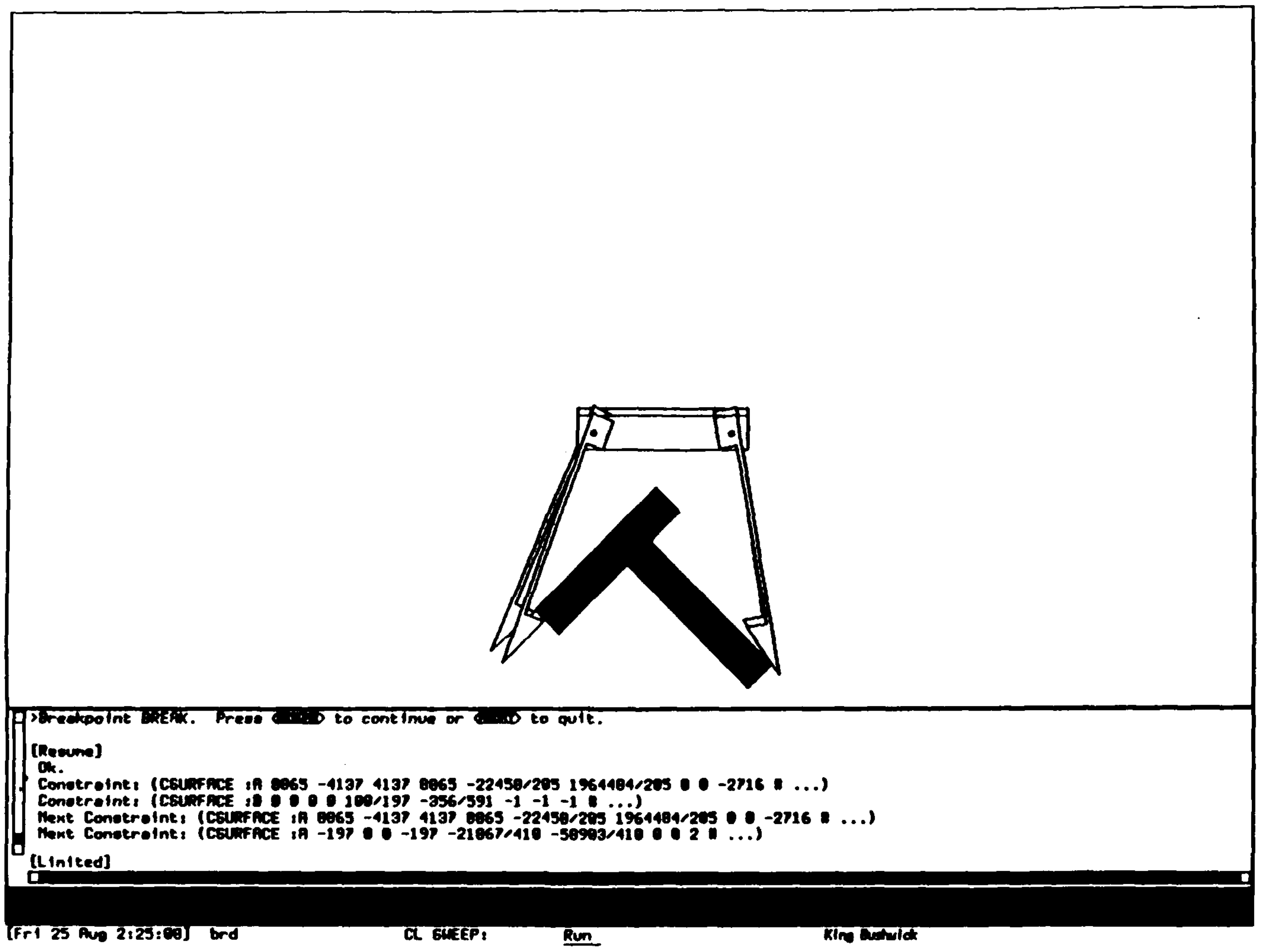
```
[Fri 25 Aug 2:17:42]  brd              CL SWEEP:       Run                    King Bushwick
```

fig 2c



```
• Control-R Return
Return a value from the function SIMULATE-TCC.
New value to return: nil
 Ok.
 Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964404/205 0 0 -2716 0 ...)
 Constraint: (CSURFACE :B 0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)
 Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964404/205 0 0 -2716 0 ...)
 Next Constraint: (CSURFACE :B 0 0 0 0 0 100/197 -356/591 -1 -1 -1 0 ...)

[Limited]
[Fri 25 Aug 2:23:39]  brd                CL SWEEP:      Run                King Bushwick
```

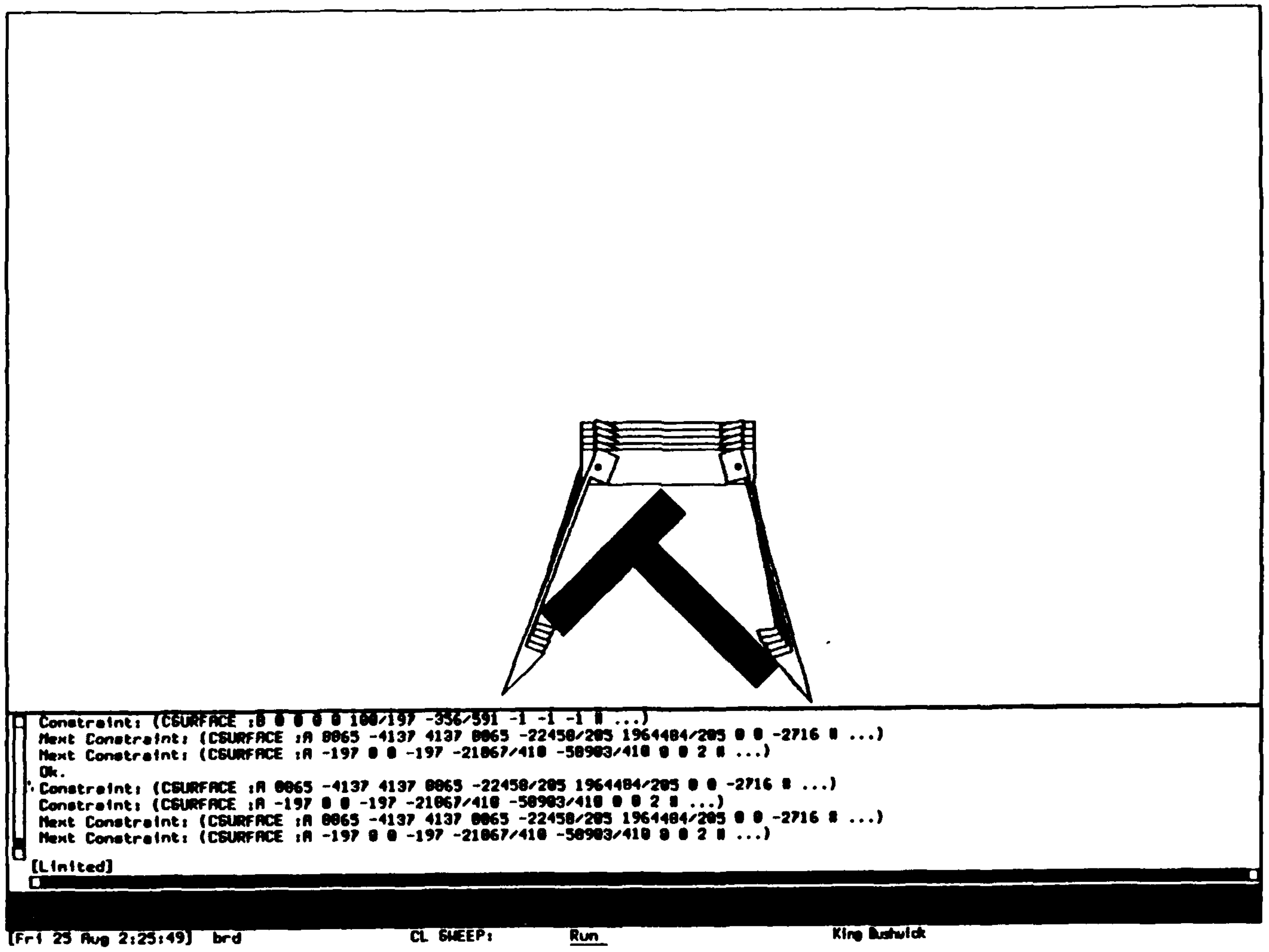
fig 2d



```
>Breakpoint BREAK.  Press [illegible] to continue or [illegible] to quit.

[Resume]
 Ok.
 Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 0 ...)
 Constraint: (CSURFACE :0 0 0 0 0 100/197 -356/591 -1 -1 -1 0 ...)
 Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 0 ...)
 Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -50903/410 0 0 2 0 ...)

[Limited]
```

```
[Fri 25 Aug 2:25:00]  brd            CL SWEEP:      Run              King Bushwick
```

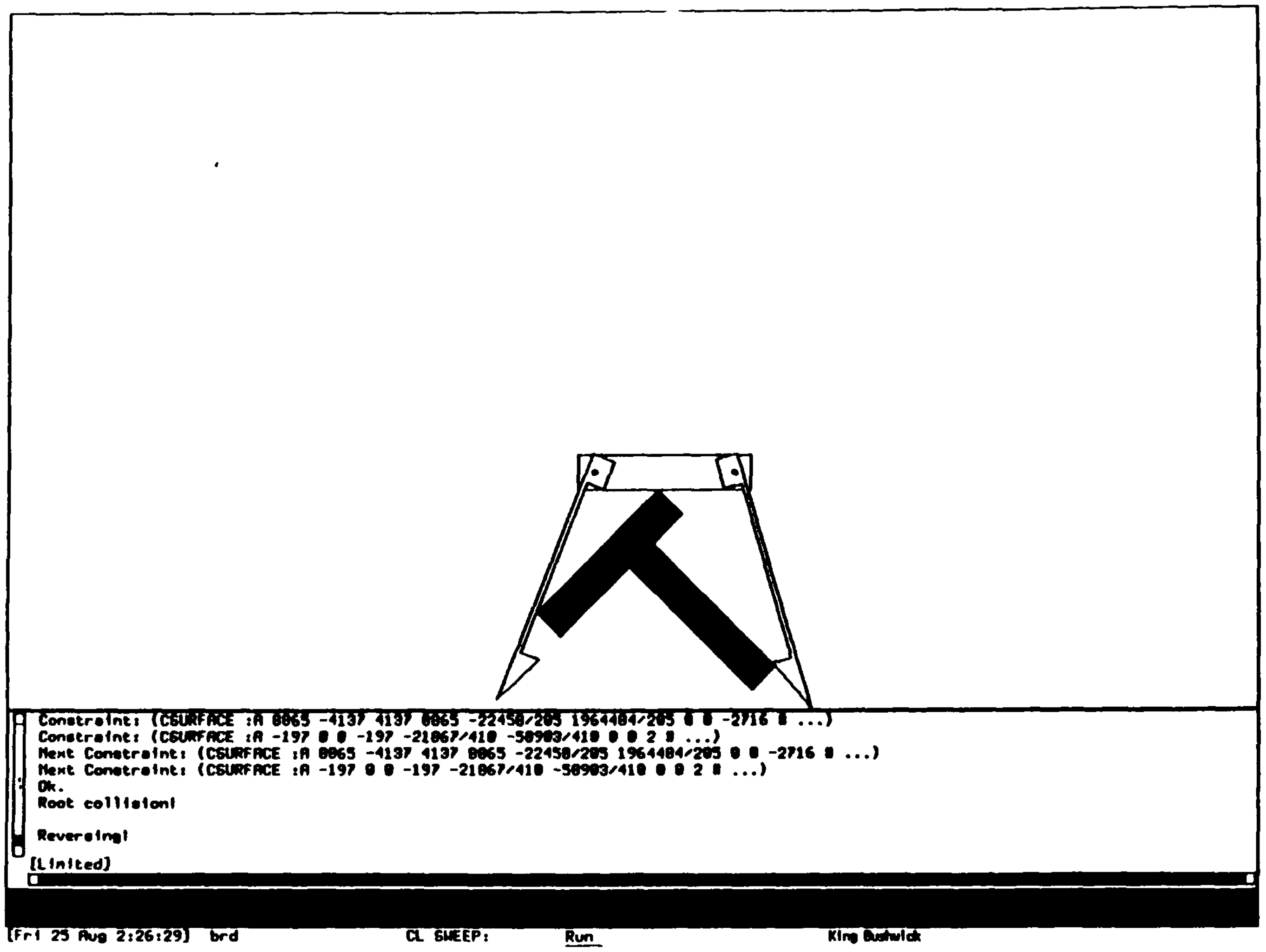
fig 2e



```
Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 0 ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 0 ...)
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 0 ...)
Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 0 ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 0 ...)
[Limited]
```

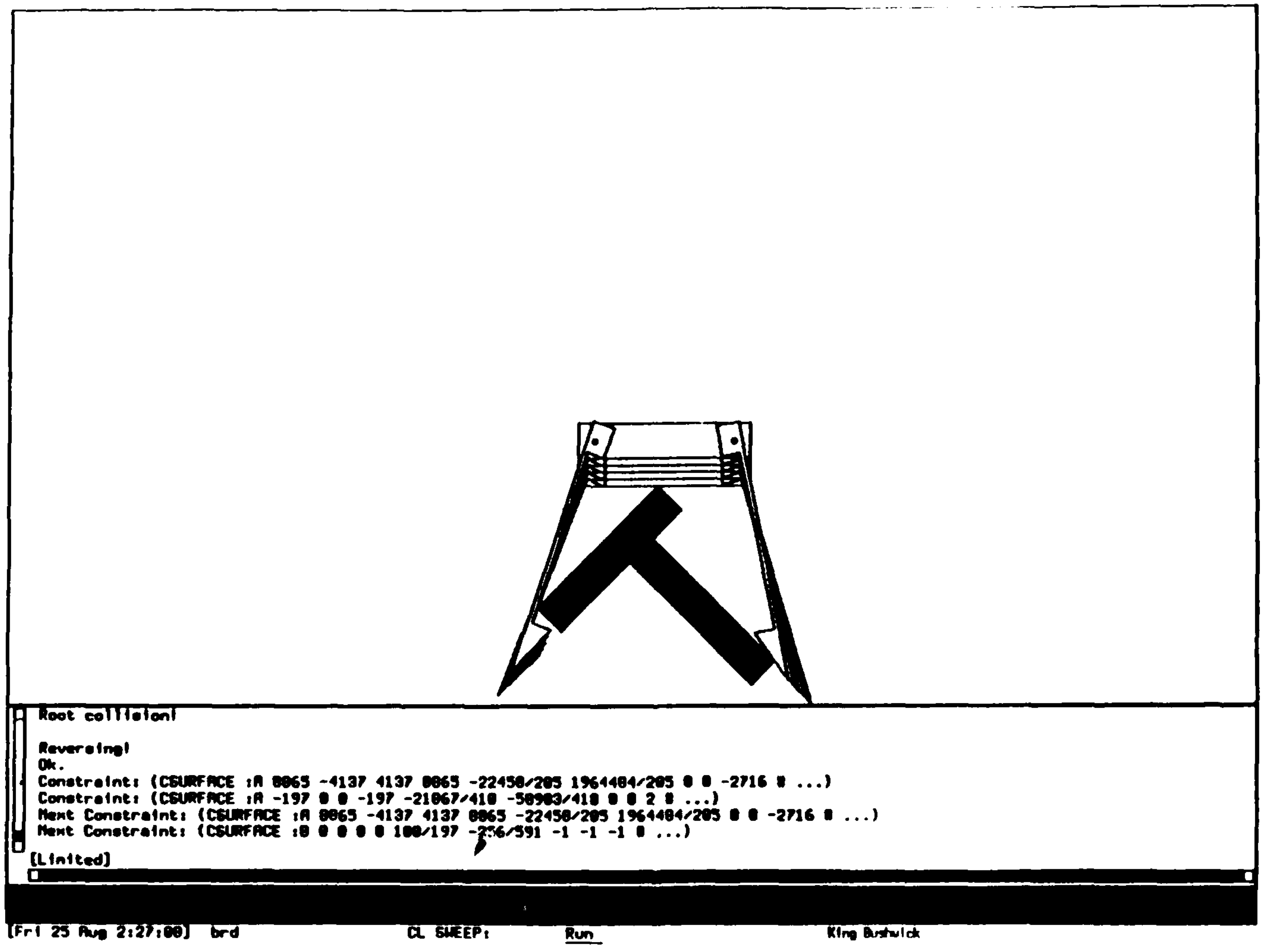
```
[Fri 25 Aug 2:25:49]  brd            CL SWEEP:      Run                 King Bushwick
```

Fig 2f



```
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 0 ...)
Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 0 ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 0 ...)
Ok.
Root collision!

Reversing!
[Limited]
```

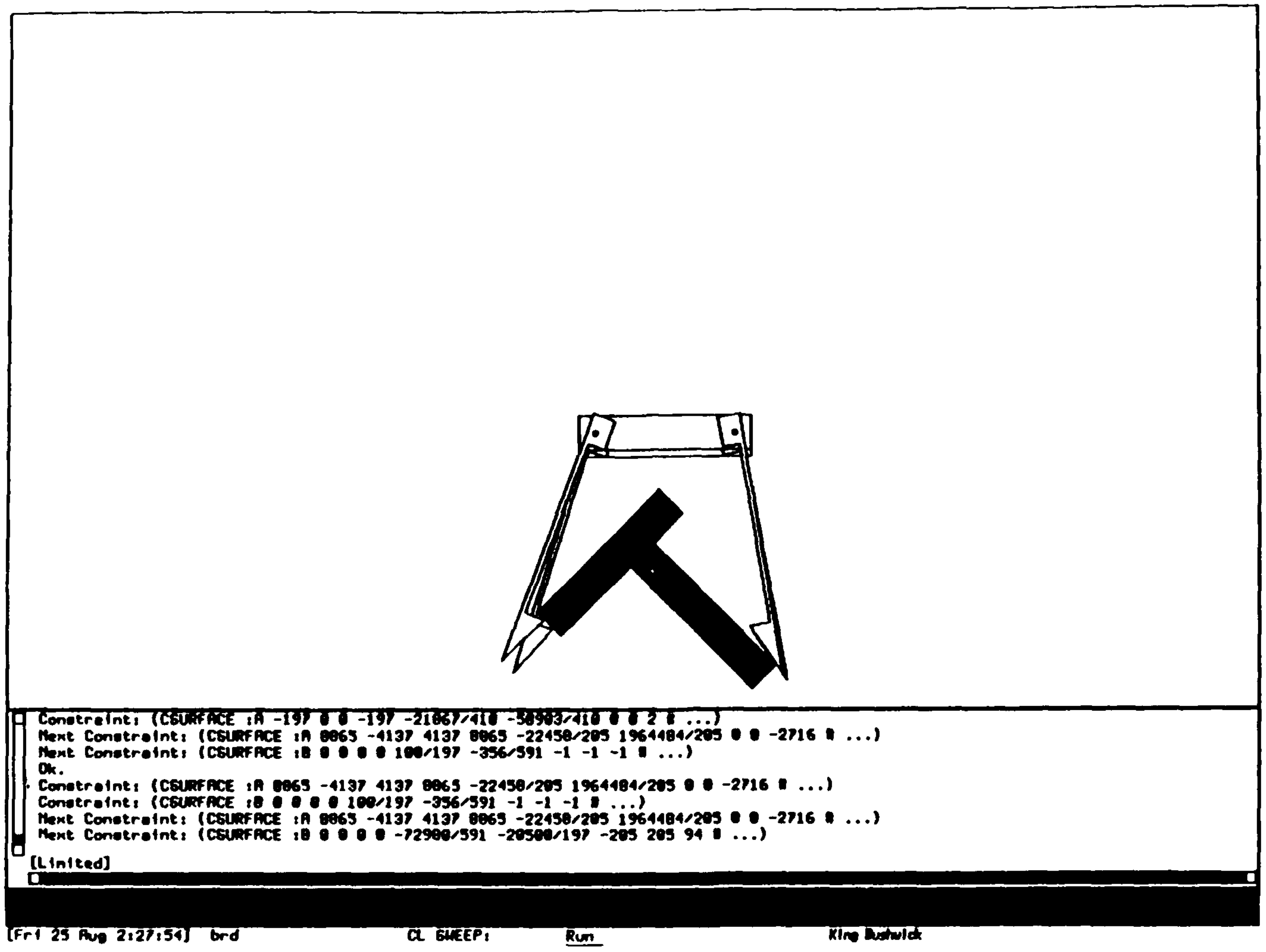
fig 2g



```
Root collision!

Reversing!
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964404/205 0 0 -2716 0 ...)
Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -50903/410 0 0 2 0 ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964404/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 0 100/197 -256/591 -1 -1 -1 0 ...)
[Limited]
```

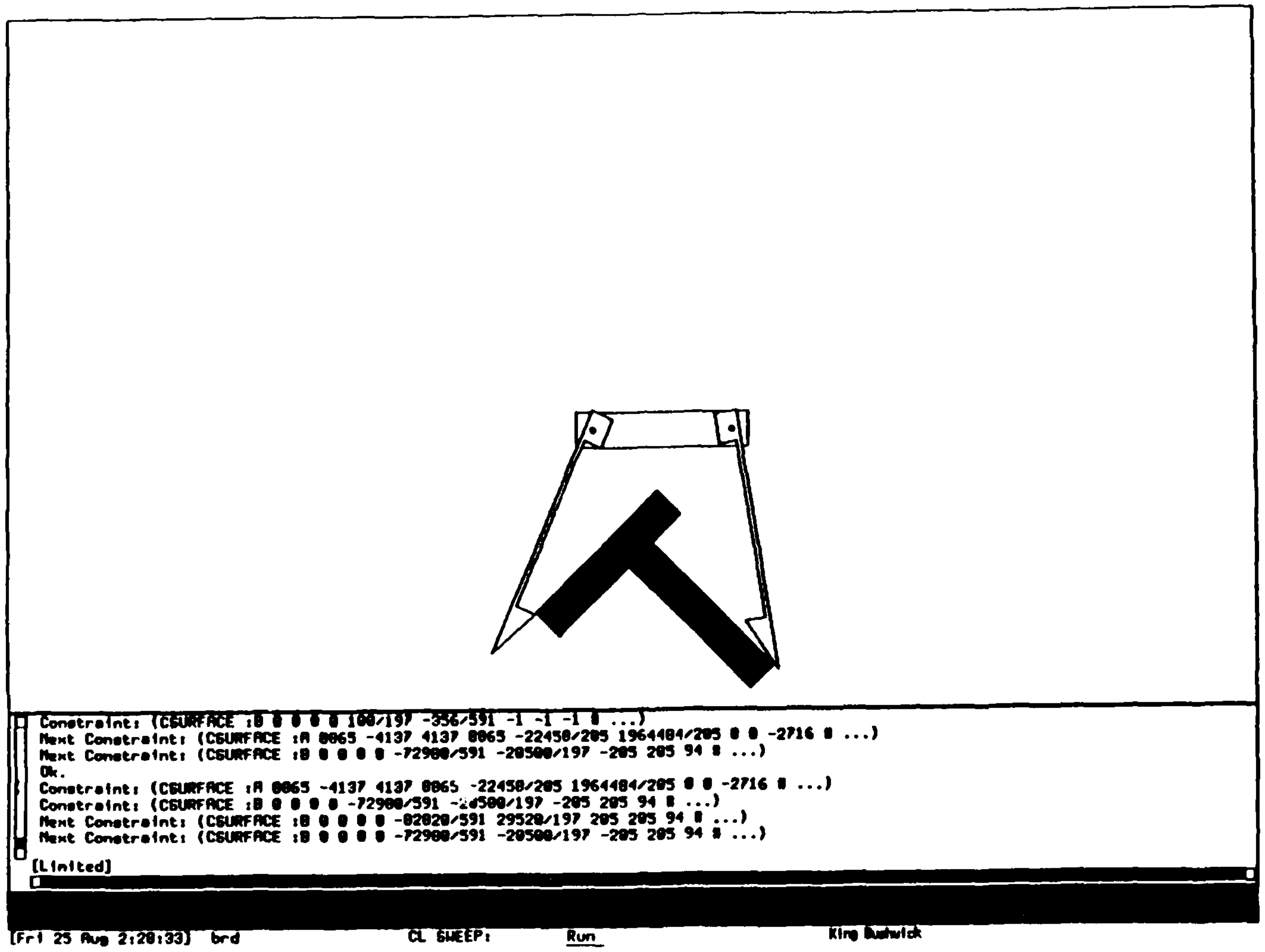
```
[Fri 25 Aug 2:27:00]   brd                CL SWEEP:     Run                King Bushwick
```

fig 2h



```
Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 0 ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 0 ...)
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 0 ...)
Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 0 ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)

[Limited]
[Fri 25 Aug 2:27:54]  brd            CL SWEEP:      Run                    King Bushwick
```

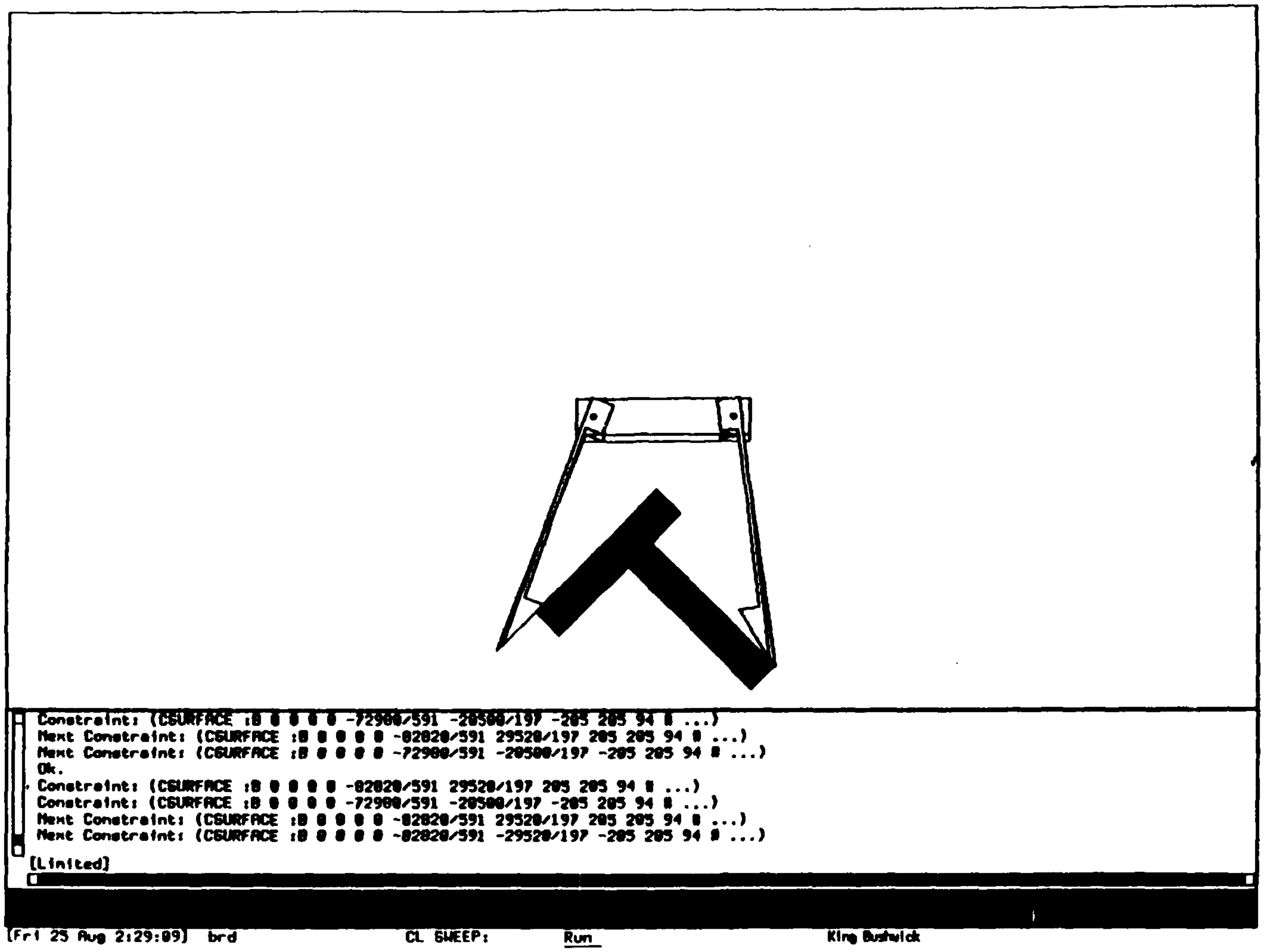
fig 2i



```
Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 0 ...)
Next Constraint: (CSURFACE :A 0065 -4137 4137 0065 -22458/205 1964404/205 0 0 -2716 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)
Ok.
Constraint: (CSURFACE :A 0065 -4137 4137 0065 -22458/205 1964404/205 0 0 -2716 0 ...)
Constraint: (CSURFACE :B 0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)

[Limited]
```

```
[Fri 25 Aug 2:20:33]  brd            CL SWEEP:      Run              King Bushwick
```

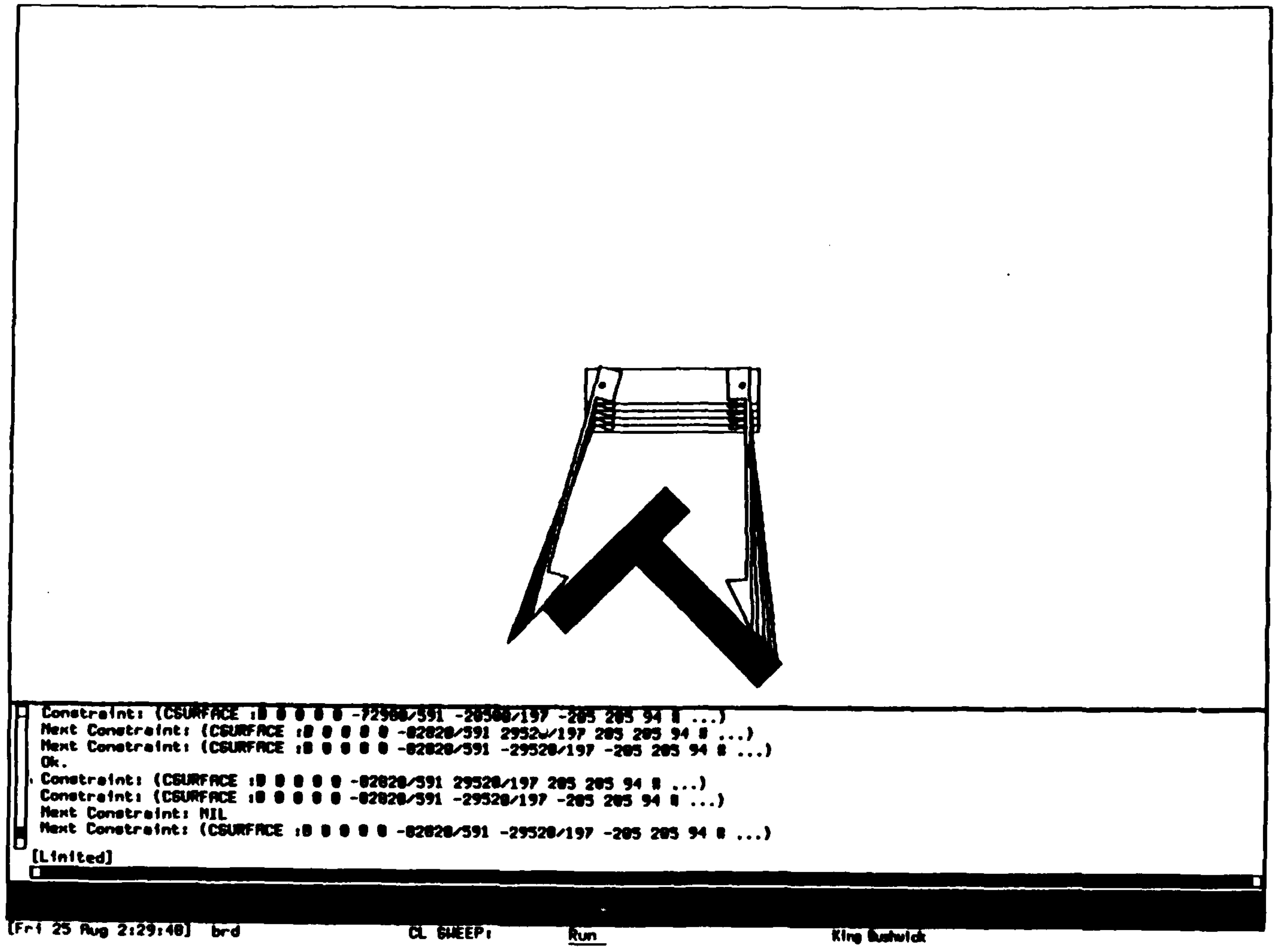
fig 2j



```
Constraint: (CSURFACE :0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)
Ok.
Constraint: (CSURFACE :0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Constraint: (CSURFACE :0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)

[Limited]
```

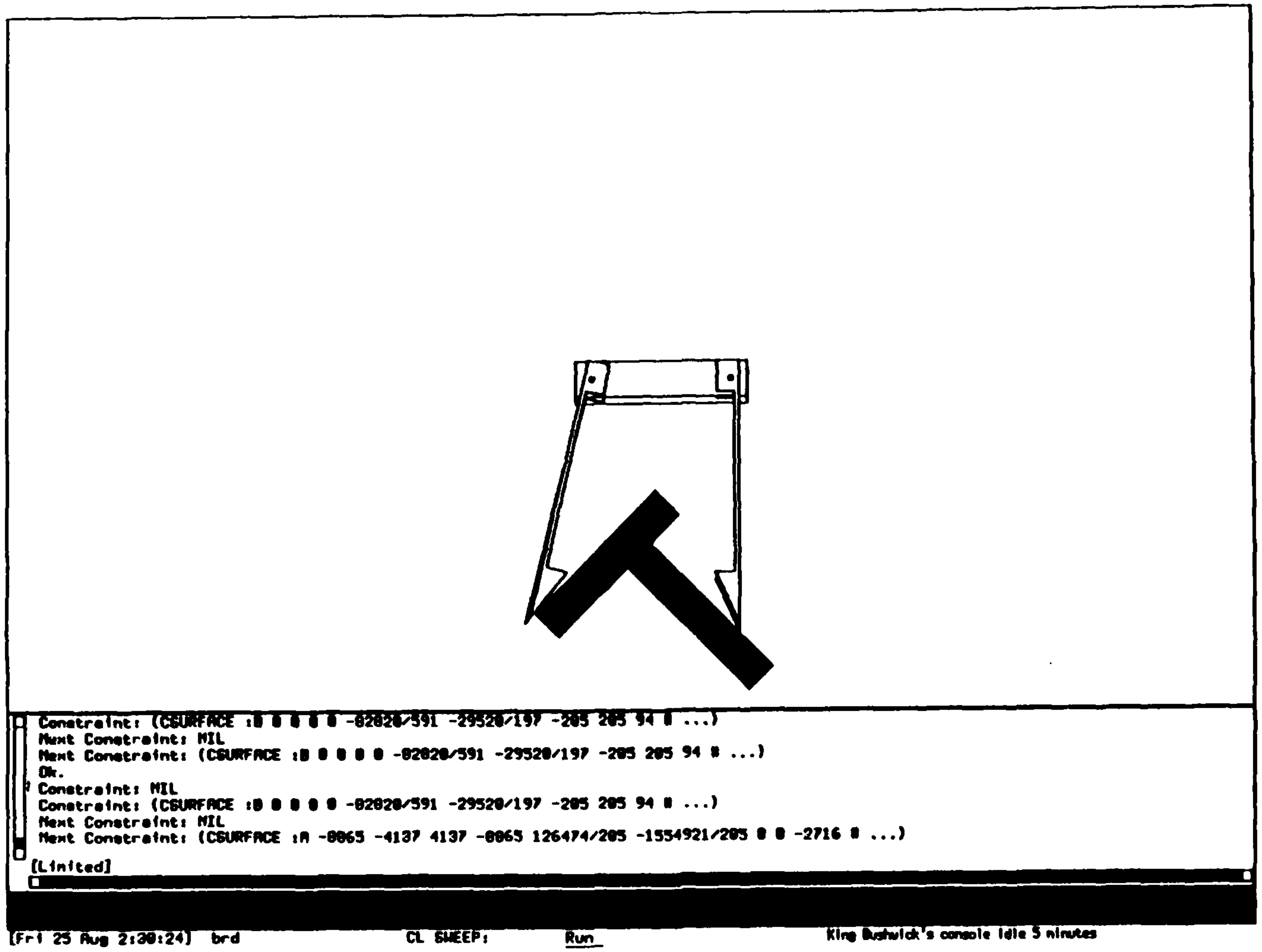
```
[Fri 25 Aug 2:29:09]  brd          CL SWEEP:     Run              King Bushwick
```

fig 2k



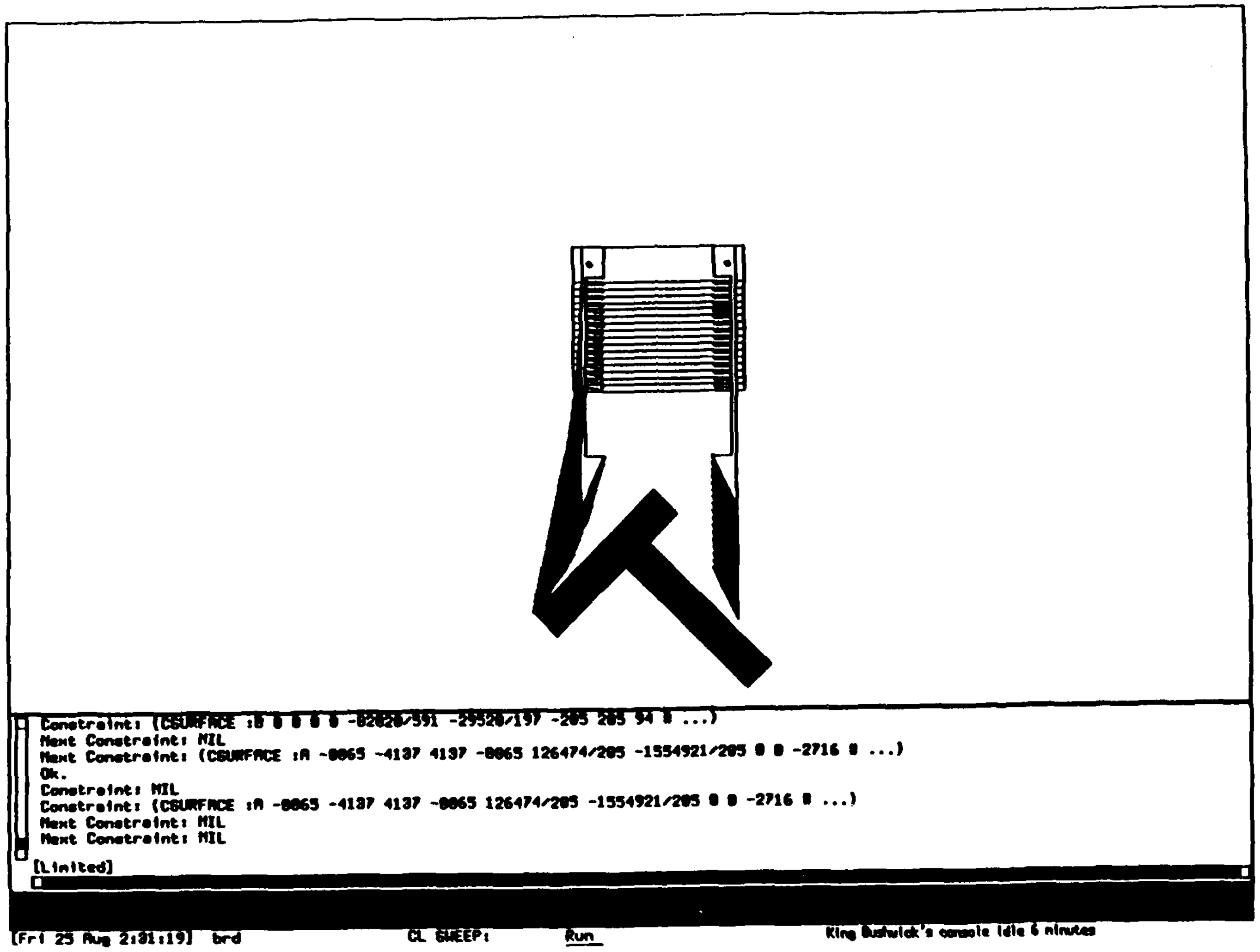
```
Constraint: (CSURFACE :0 0 0 0 0 -72900/591 -20500/197 -205 205 94 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 2952ω/197 205 205 94 0 ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)
Ok.
Constraint: (CSURFACE :0 0 0 0 0 -82020/591 29520/197 205 205 94 0 ...)
Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)
[Limited]
```

fig 2l



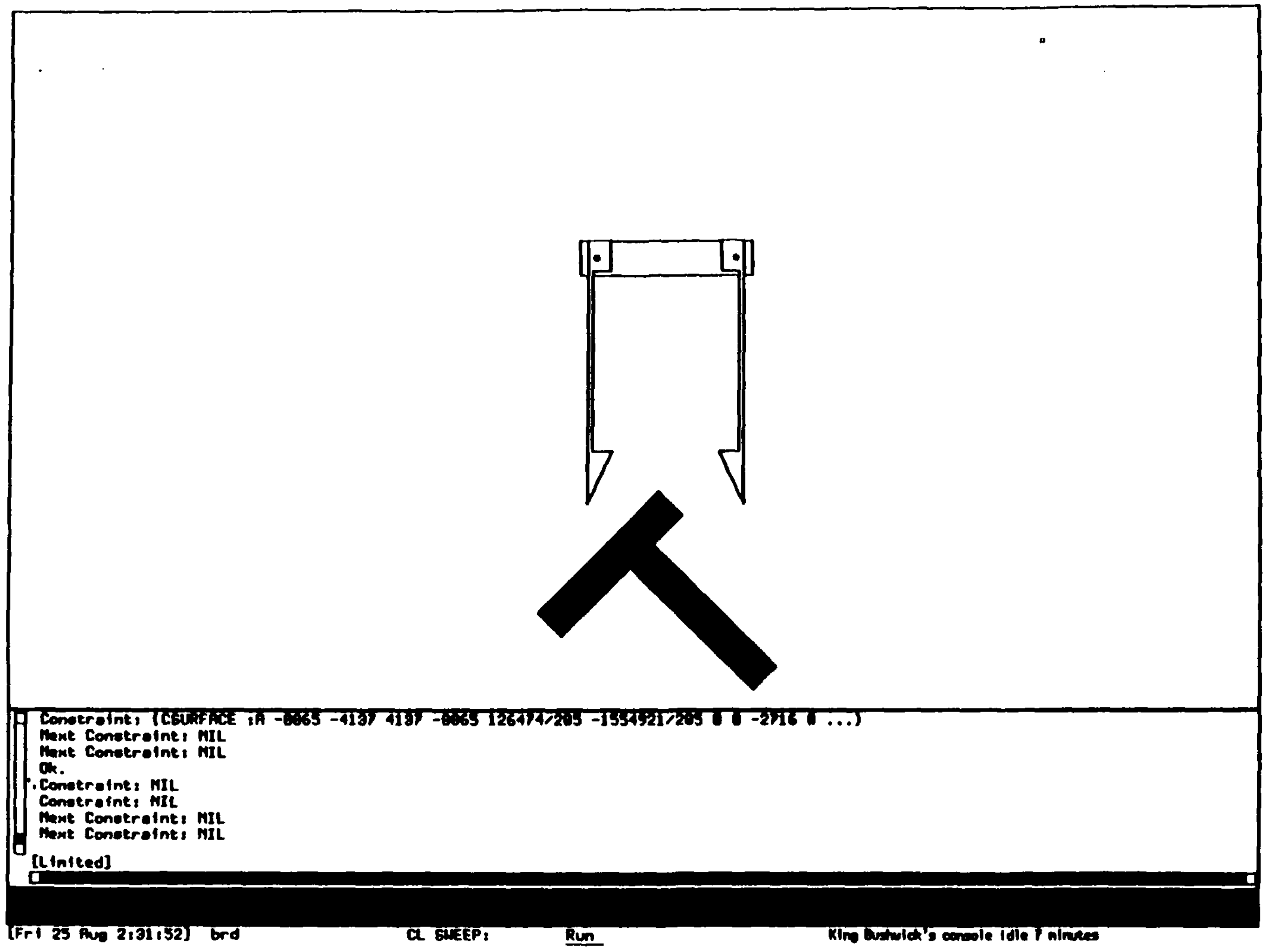
```
Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 0 ...)

[Limited]
[Fri 25 Aug 2:30:24]  brd              CL SWEEP:       Run              King Bushwick's console idle 5 minutes
```

fig 2m



```
Constraint: (CSURFACE :0 0 0 0 0 -82820/591 -29520/197 -205 205 94 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 0 ...)
Next Constraint: NIL
Next Constraint: NIL
[Limited]
[Fri 25 Aug 2:31:19]  brd           CL SWEEP:      Run           King Bushwick's console idle 6 minutes
```

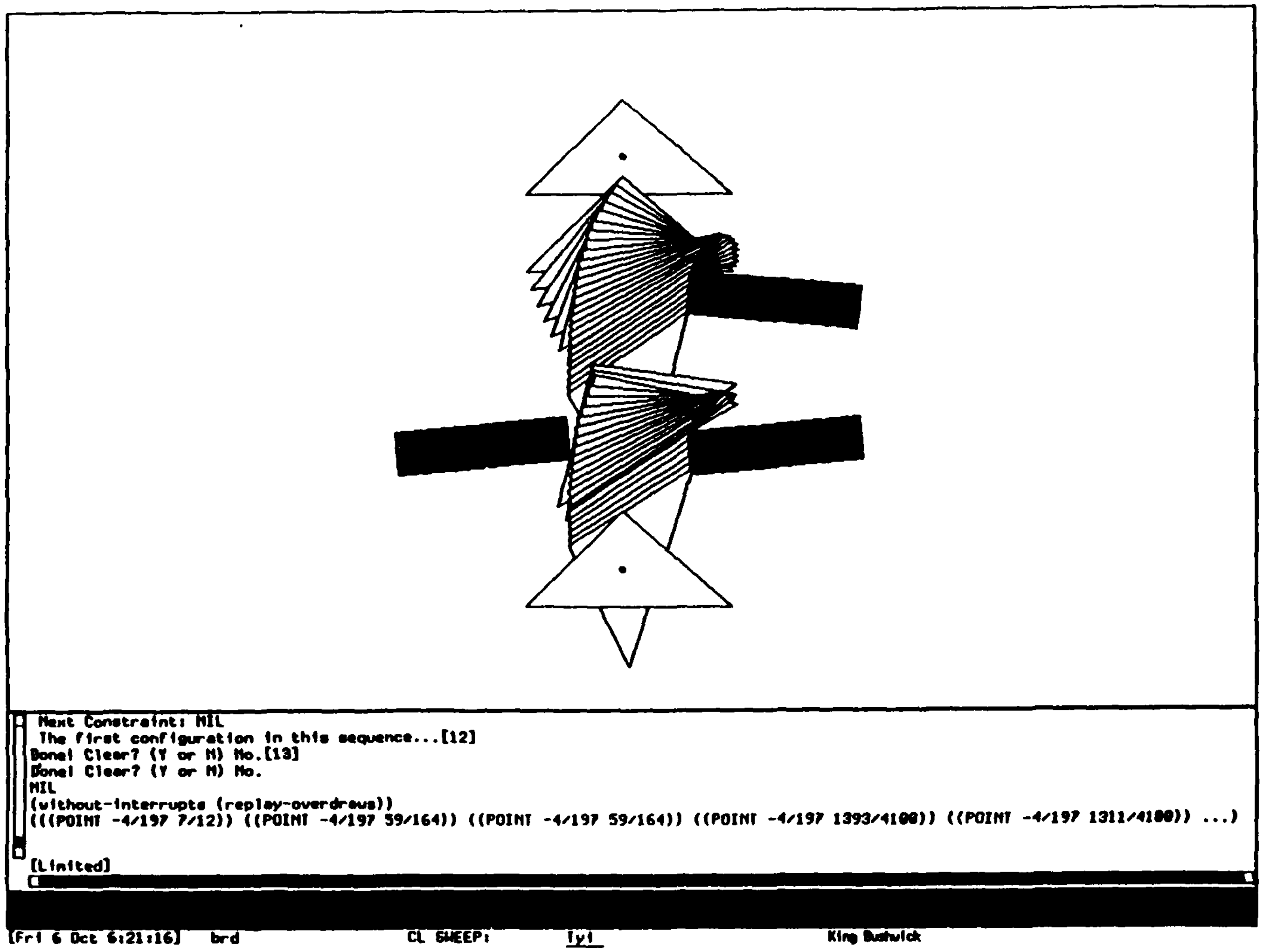
fig 2n



```
Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 0 ...)
Next Constraint: NIL
Next Constraint: NIL
Ok.
.Constraint: NIL
Constraint: NIL
Next Constraint: NIL
Next Constraint: NIL

[Limited]
```

```
[Fri 25 Aug 2:31:52]  brd            CL SWEEP:       Run                    King Bushwick's console idle 7 minutes
```

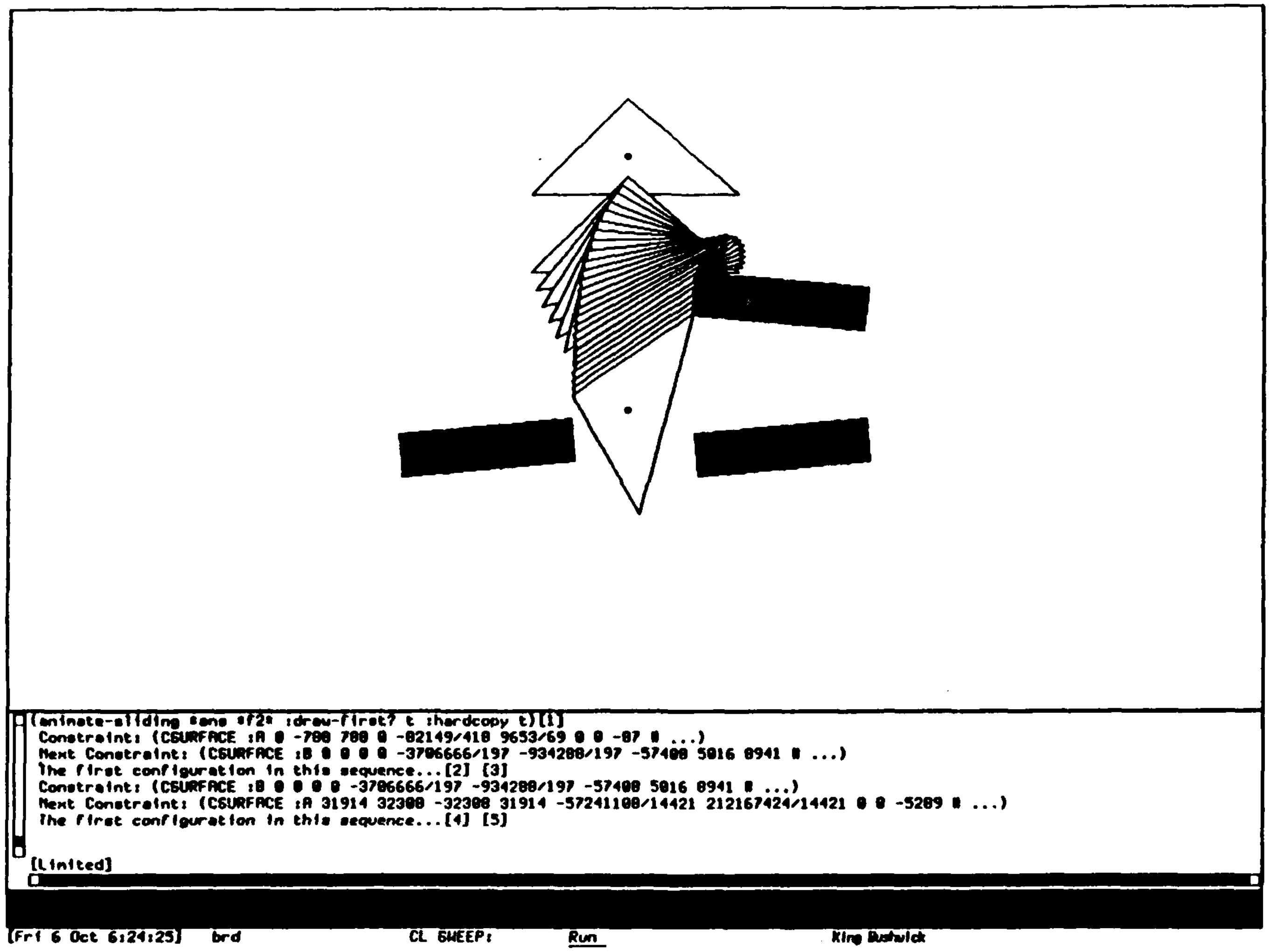
fig 2-0



```
 Next Constraint: NIL
 The first configuration in this sequence...[12]
Done! Clear? (Y or N) No.[13]
Done! Clear? (Y or N) No.
NIL
(without-interrupts (replay-overdraws))
(((POINT -4/197 7/12)) ((POINT -4/197 59/164)) ((POINT -4/197 59/164)) ((POINT -4/197 1393/4100)) ((POINT -4/197 1311/4100)) ...)

[Limited]
```

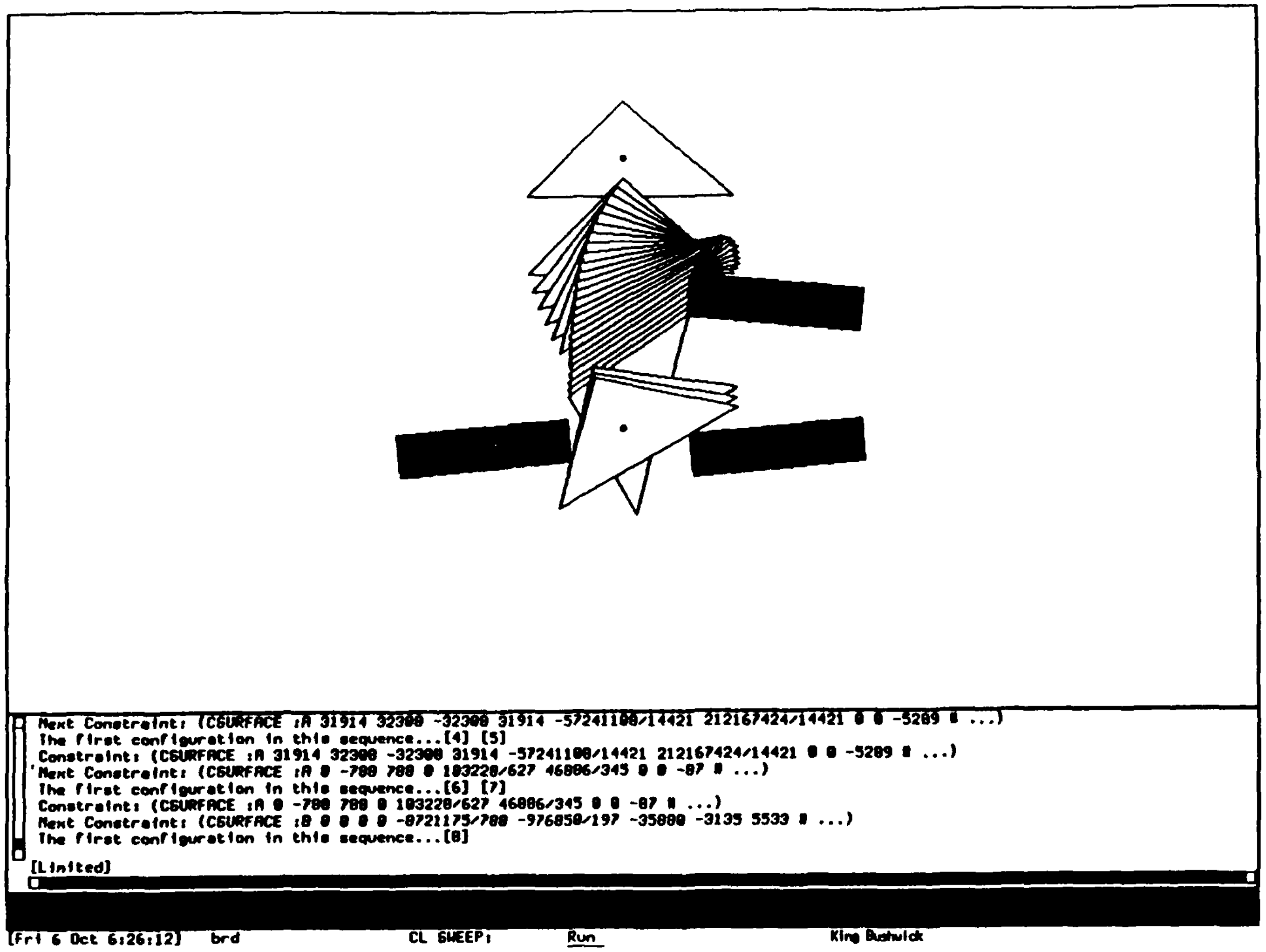
```
[Fri 6 Oct 6:21:16]   brd          CL SWEEP:     Tyi               King Bushwick
```

3a



```
(animate-sliding *ans *f2* :draw-first? t :hardcopy t)[1]
 Constraint: (CSURFACE :A 0 -700 700 0 -82149/410 9653/69 0 0 -87 0 ...)
 Next Constraint: (CSURFACE :B 0 0 0 0 -3706666/197 -934280/197 -57400 5016 0941 0 ...)
 The first configuration in this sequence...[2] [3]
 Constraint: (CSURFACE :B 0 0 0 0 -3706666/197 -934280/197 -57400 5016 0941 0 ...)
 Next Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 0 ...)
 The first configuration in this sequence...[4] [5]

[Linited]
```

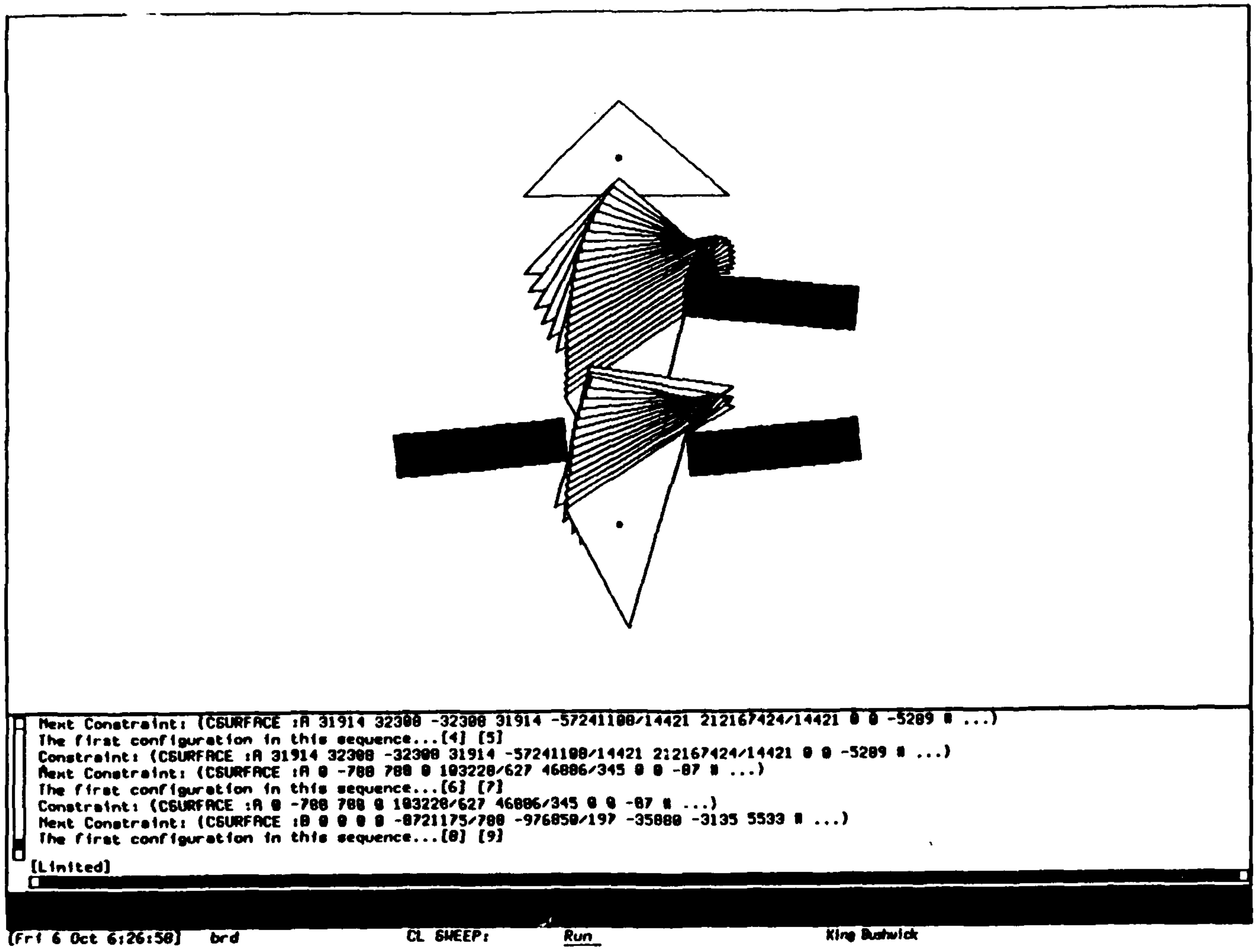
3b



```
Next Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 # ...)
The first configuration in this sequence...[4] [5]
Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 # ...)
Next Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46806/345 0 0 -87 # ...)
The first configuration in this sequence...[6] [7]
Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46806/345 0 0 -87 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -8721175/700 -976850/197 -35000 -3135 5533 # ...)
The first configuration in this sequence...[8]

[Linited]
```

```
[Fri 6 Oct 6:26:12]    brd              CL SWEEP:      Run                King Bushwick
```

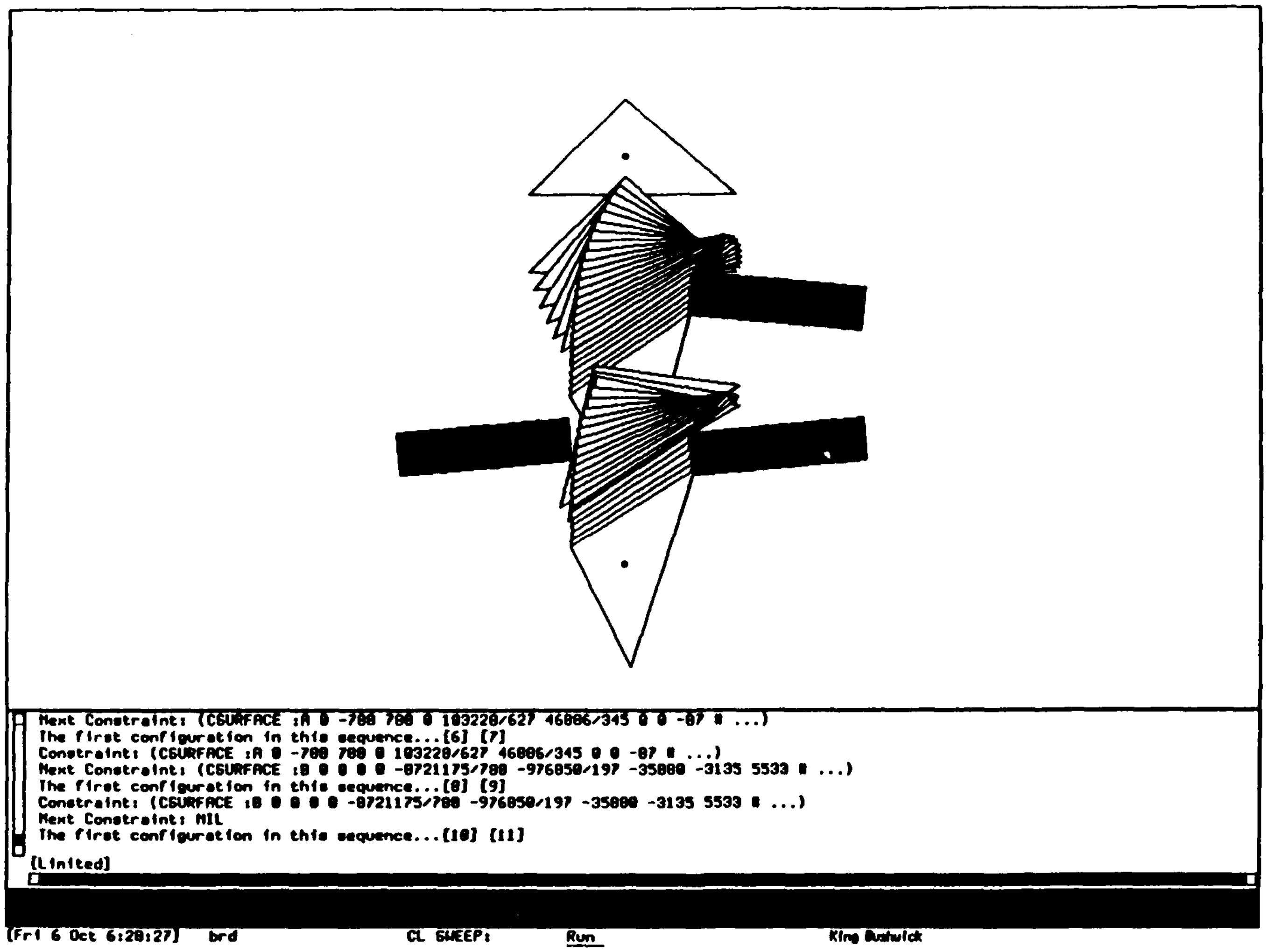
3c



```
Next Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 0 ...)
The first configuration in this sequence...[4] [5]
Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 0 ...)
Next Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46006/345 0 0 -87 0 ...)
The first configuration in this sequence...[6] [7]
Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46006/345 0 0 -87 0 ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -8721175/700 -976850/197 -35800 -3135 5533 0 ...)
The first configuration in this sequence...[8] [9]

[Limited]
```

```
[Fri 6 Oct 6:26:58]    brd          CL SWEEP:      Run                King Bushwick
```

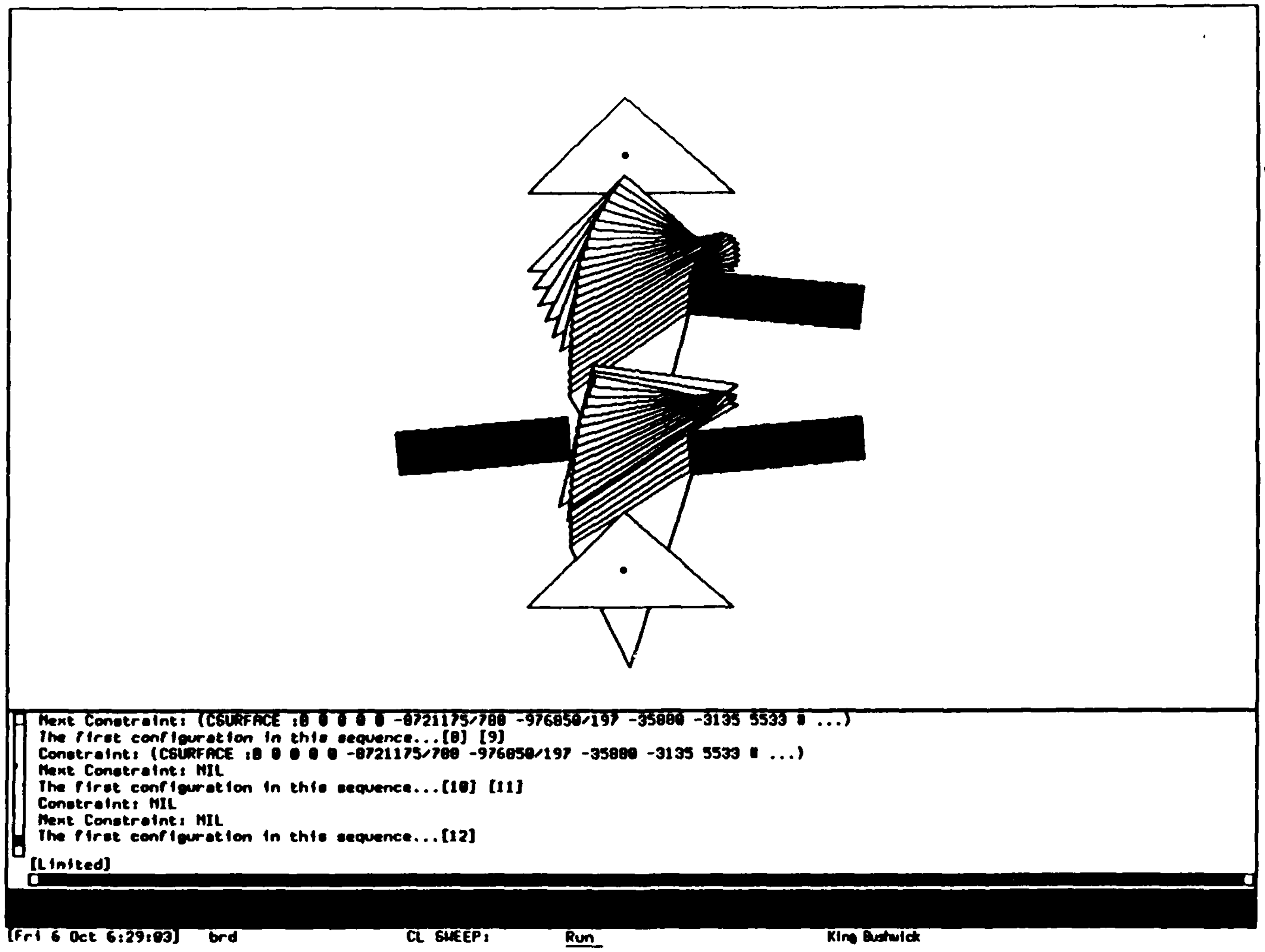
3d



```
Next Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46006/345 0 0 -07 # ...)
The first configuration in this sequence...[6] [7]
Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46006/345 0 0 -07 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -0721175/700 -976050/197 -35000 -3135 5533 # ...)
The first configuration in this sequence...[8] [9]
Constraint: (CSURFACE :B 0 0 0 0 -0721175/700 -976050/197 -35000 -3135 5533 # ...)
Next Constraint: NIL
The first configuration in this sequence...[10] [11]

[Linited]
```

```
(Fri 6 Oct 6:20:27]    brd            CL SWEEP:       Run              King Bushwick
```

3e

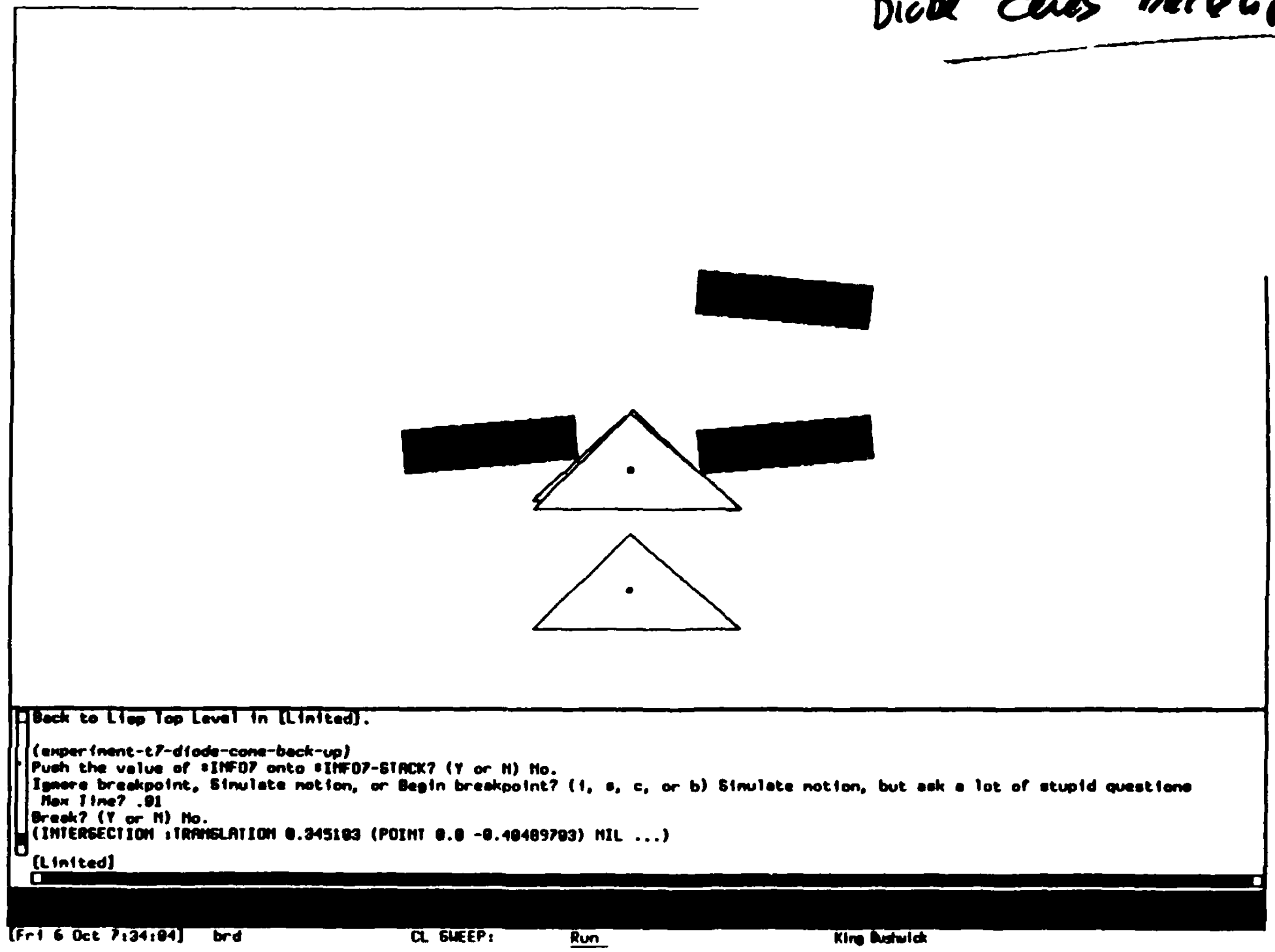


```
Next Constraint: (CSURFACE :0 0 0 0 0 -0721175/700 -976050/197 -35000 -3135 5533 0 ...)
The first configuration in this sequence...[8] [9]
Constraint: (CSURFACE :0 0 0 0 0 -0721175/700 -976050/197 -35000 -3135 5533 0 ...)
Next Constraint: NIL
The first configuration in this sequence...[10] [11]
Constraint: NIL
Next Constraint: NIL
The first configuration in this sequence...[12]
[Linited]
```

```
[Fri 6 Oct 6:29:03]    brd                CL SWEEP:       Run                    King Bushwick
```

3f

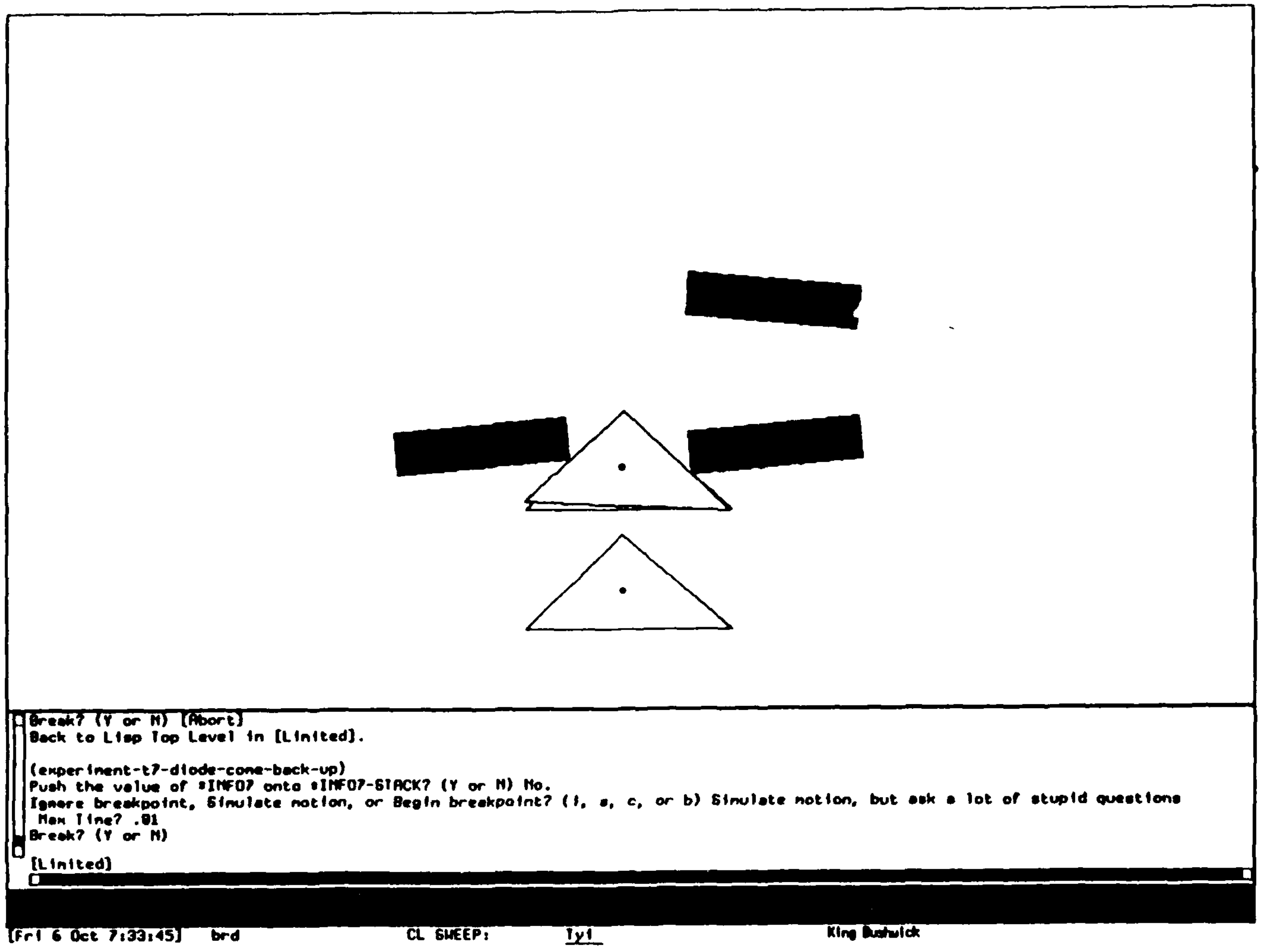
Experiment
T7
Diode cone backup



```
Back to Lisp Top Level in [Limited].

(experiment-t7-diode-cone-back-up)
Push the value of *INFO7 onto *INFO7-STACK? (Y or N) No.
Ignore breakpoint, Simulate motion, or Begin breakpoint? (I, S, C, or B) Simulate motion, but ask a lot of stupid questions
 Max Time? .01
Break? (Y or N) No.
(INTERSECTION :TRANSLATION 0.345183 (POINT 0.0 -0.40489703) NIL ...)

[Limited]
```

```
[Fri 6 Oct 7:34:04]   brd          CL SWEEP:      Run              King Bushwick
```

4a



```
Break? (Y or N) [Abort]
Back to Lisp Top Level in [Limited].

(experiment-t7-diode-cone-back-up)
Push the value of *INFO? onto *INFO?-STACK? (Y or N) No.
Ignore breakpoint, Simulate motion, or Begin breakpoint? (I, s, c, or b) Simulate motion, but ask a lot of stupid questions
 Max Time? .01
Break? (Y or N)
[Limited]
[Fri 6 Oct 7:33:45]   brd          CL SWEEP:      Ty1              King Bushwick
```

4b

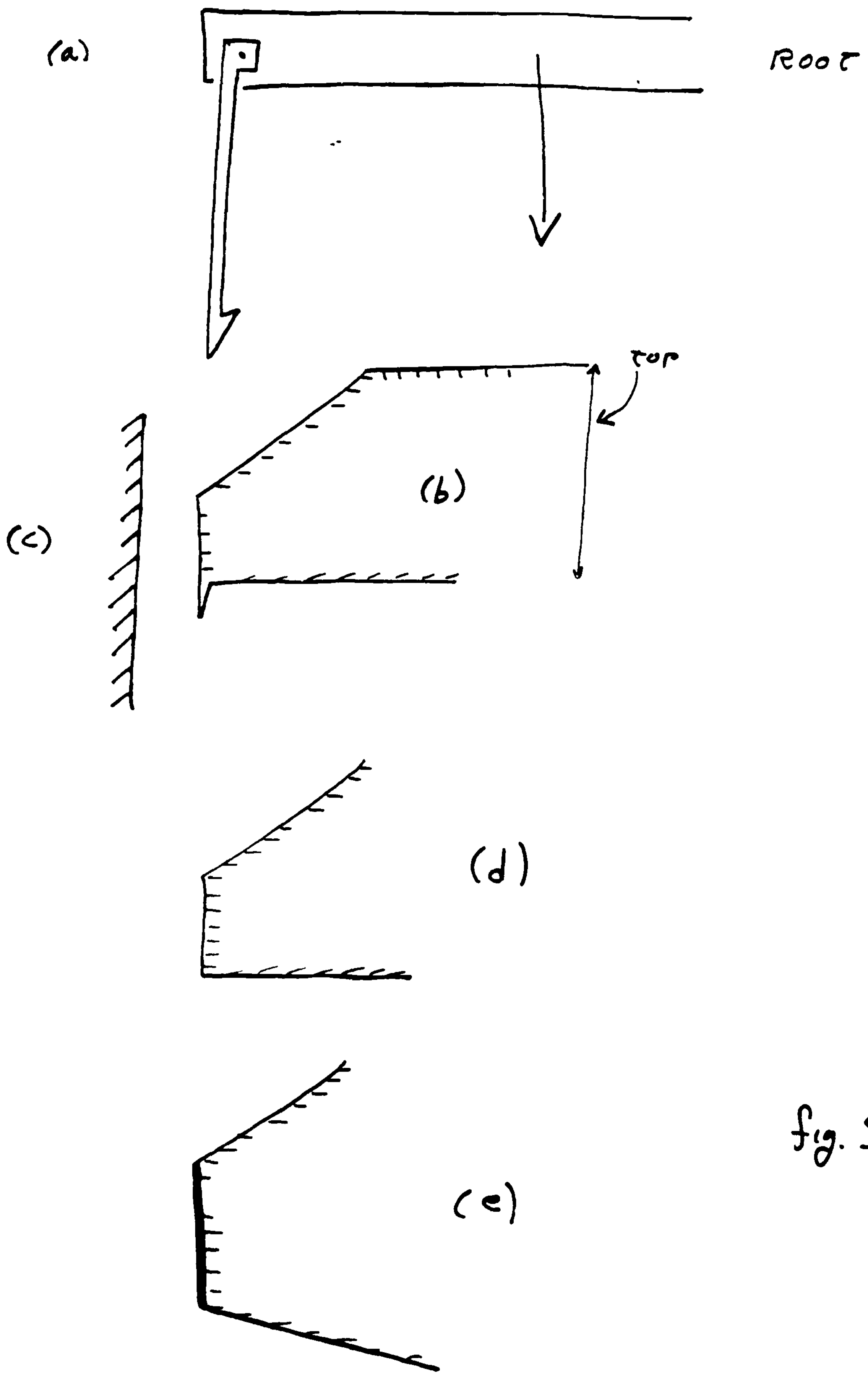
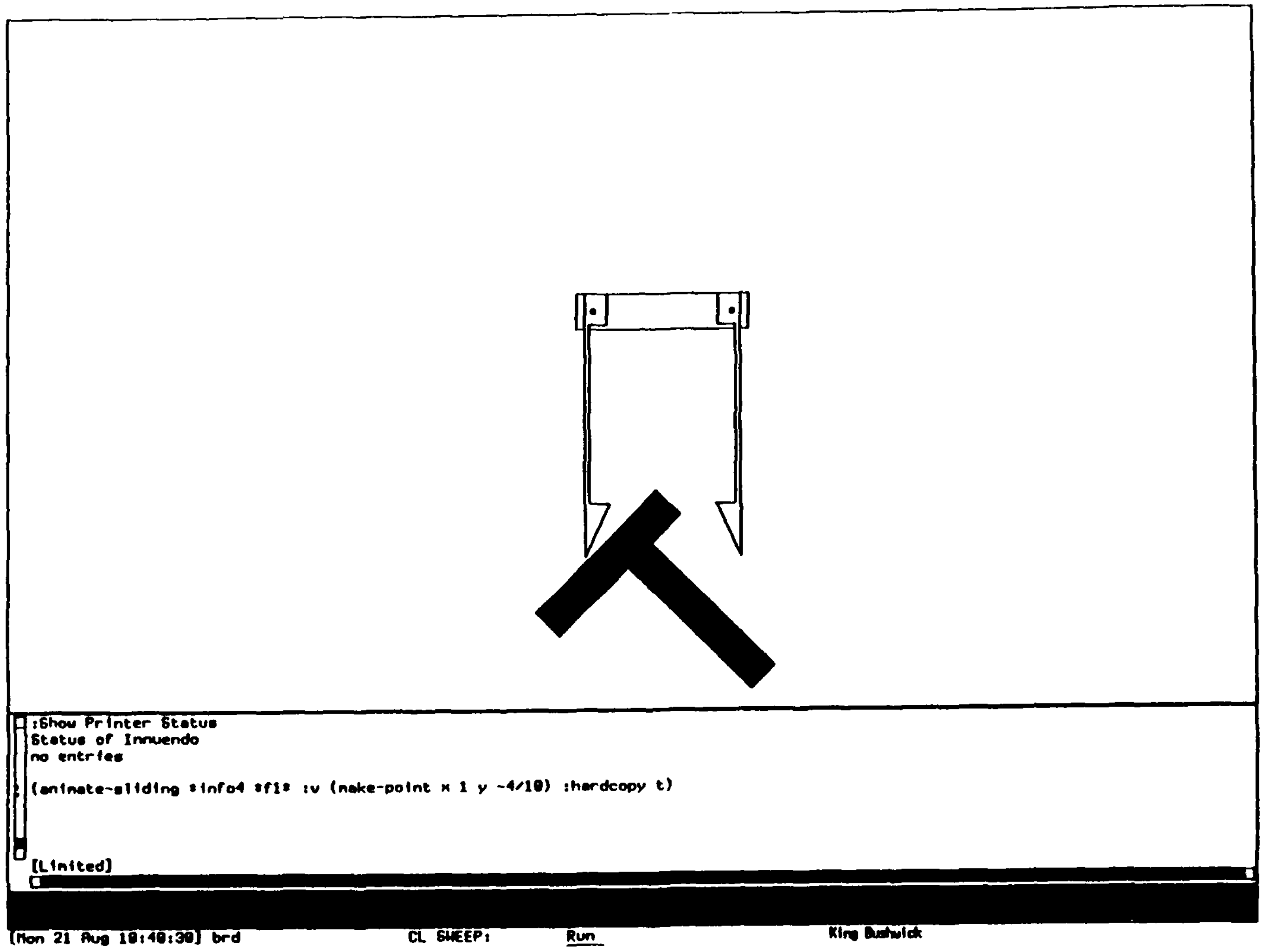
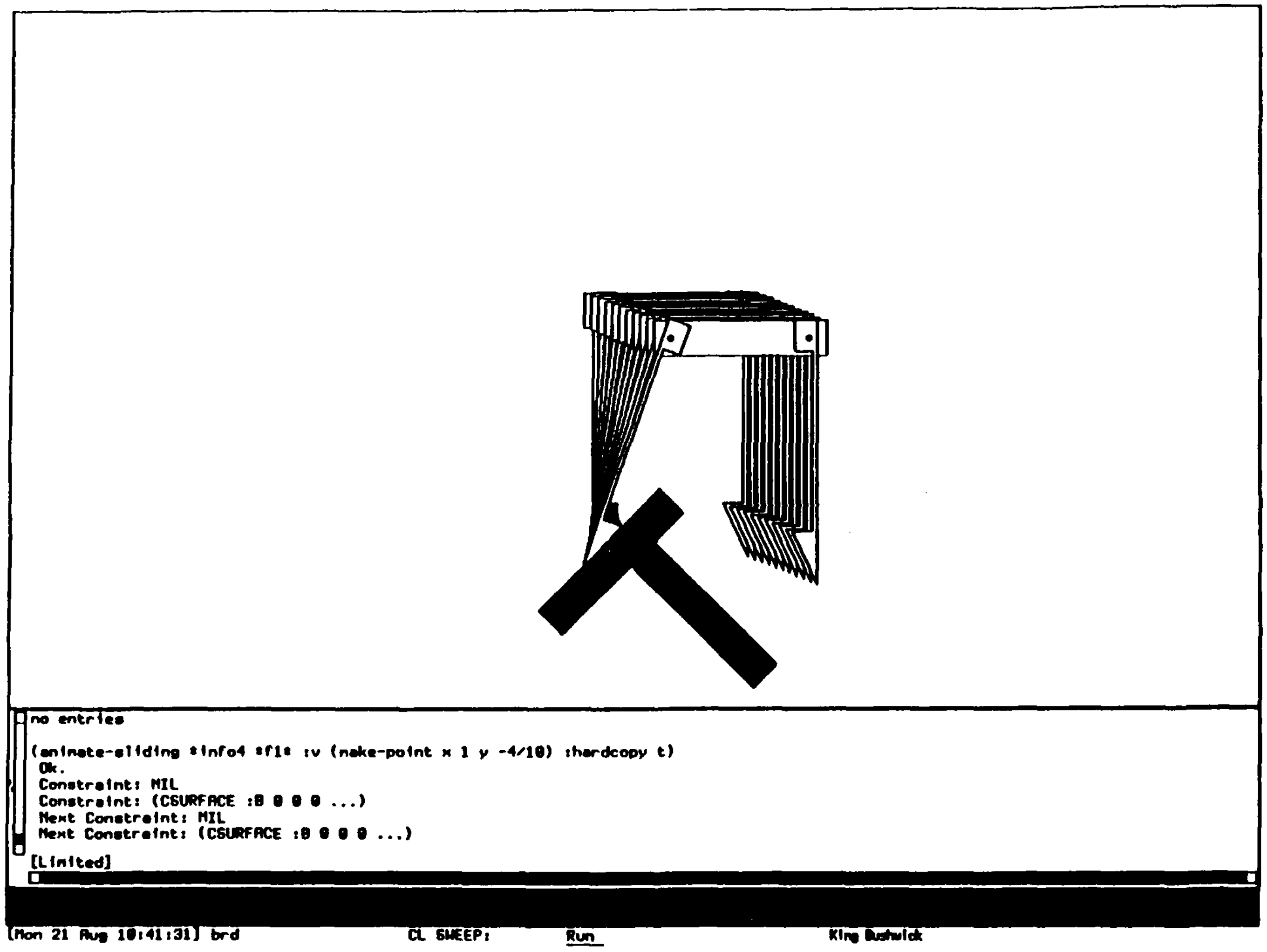


fig. 5



```
:Show Printer Status
Status of Innuendo
no entries

(animate-sliding *info4 *f1* :v (make-point x 1 y -4/10) :hardcopy t)



[Limited]
```

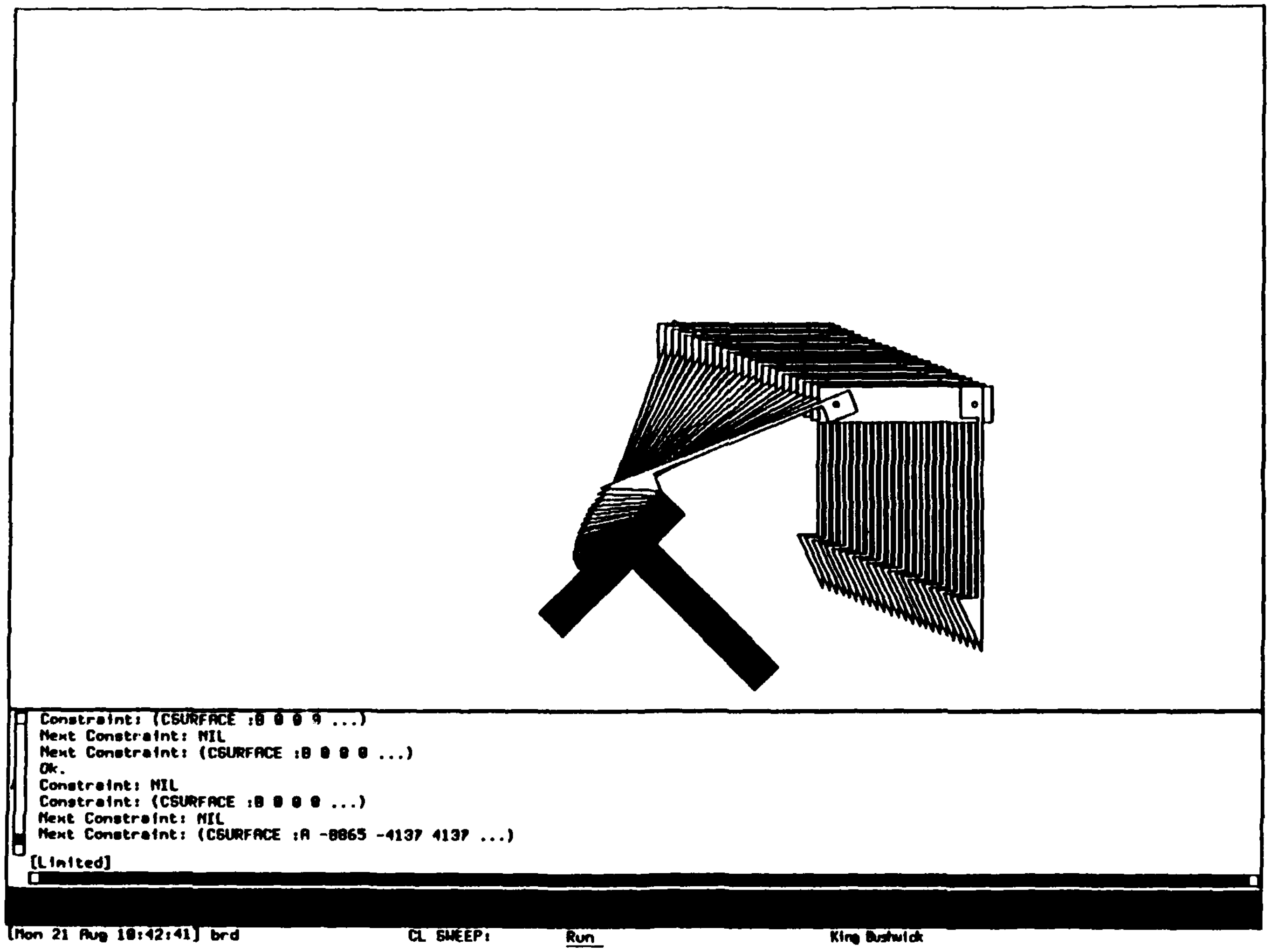
fig. 6a



```
no entries

(animate-sliding *info4 *f1* :v (make-point x 1 y -4/10) :hardcopy t)
 Ok.
 Constraint: NIL
 Constraint: (CSURFACE :0 0 0 0 ...)
 Next Constraint: NIL
 Next Constraint: (CSURFACE :0 0 0 0 ...)
[Limited]
```

```
[Mon 21 Aug 10:41:31] brd          CL SWEEP:     Run                King Bushwick
```

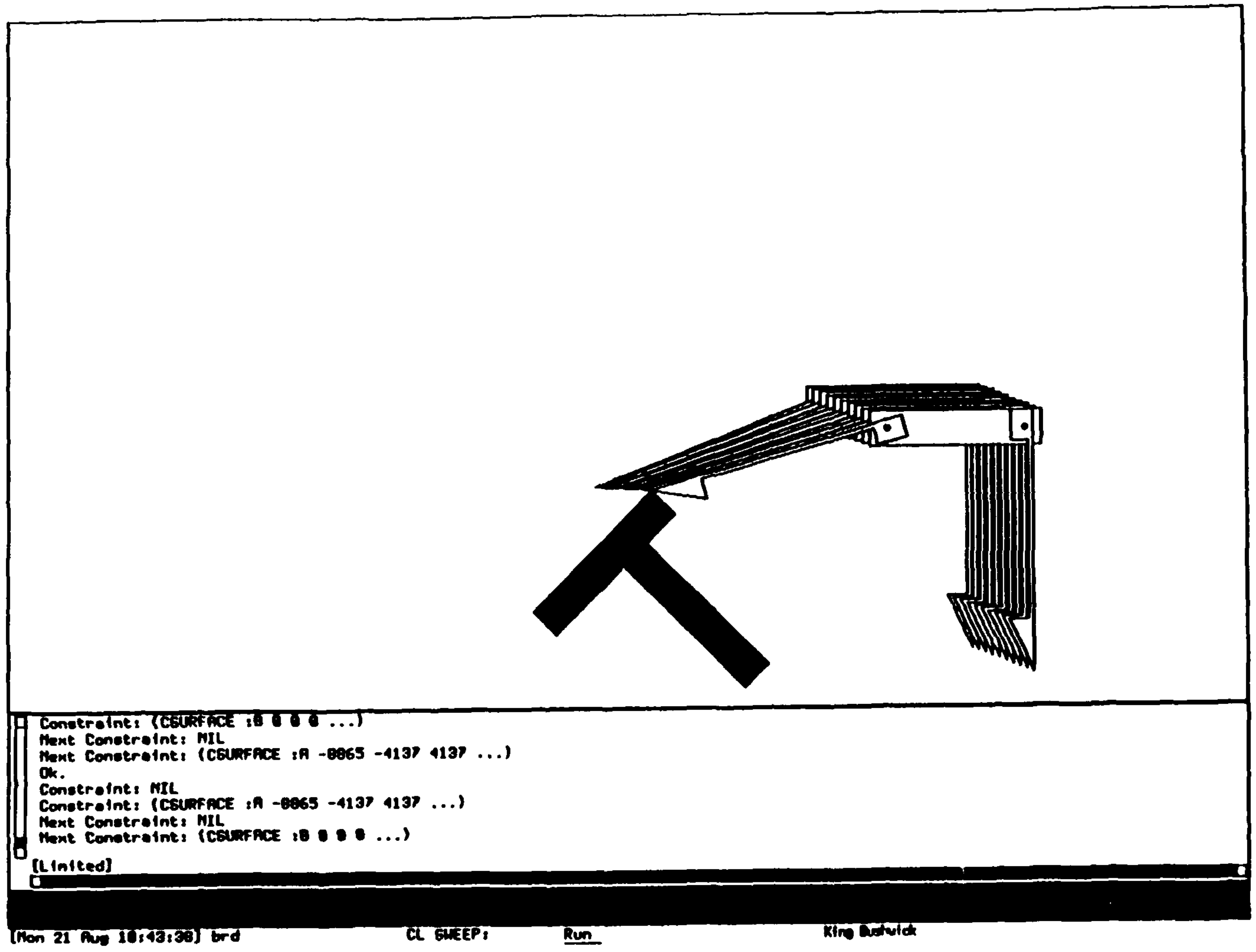
fig. 6b



```
Constraint: (CSURFACE :0 0 0 9 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8865 -4137 4137 ...)
[Limited]
```

```
[Mon 21 Aug 10:42:41] brd          CL SWEEP:     Run              King Bushwick
```

fig. 6c

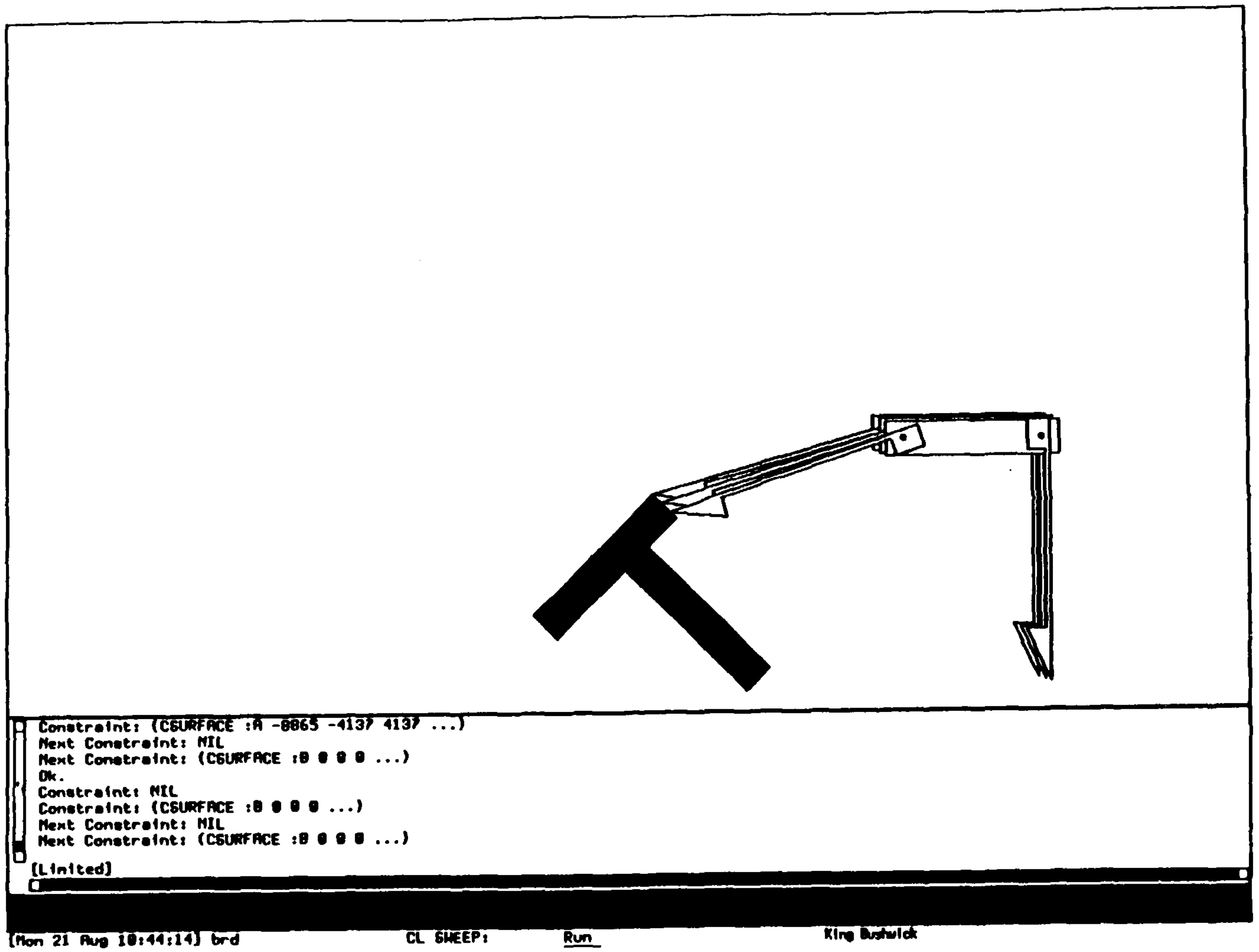


```
Constraint: (CSURFACE :B 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8865 -4137 4137 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :A -8865 -4137 4137 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :B 0 0 0 ...)
```

[Limited]

[Mon 21 Aug 10:43:38] brd   CL SWEEP: Run   King Bushwick

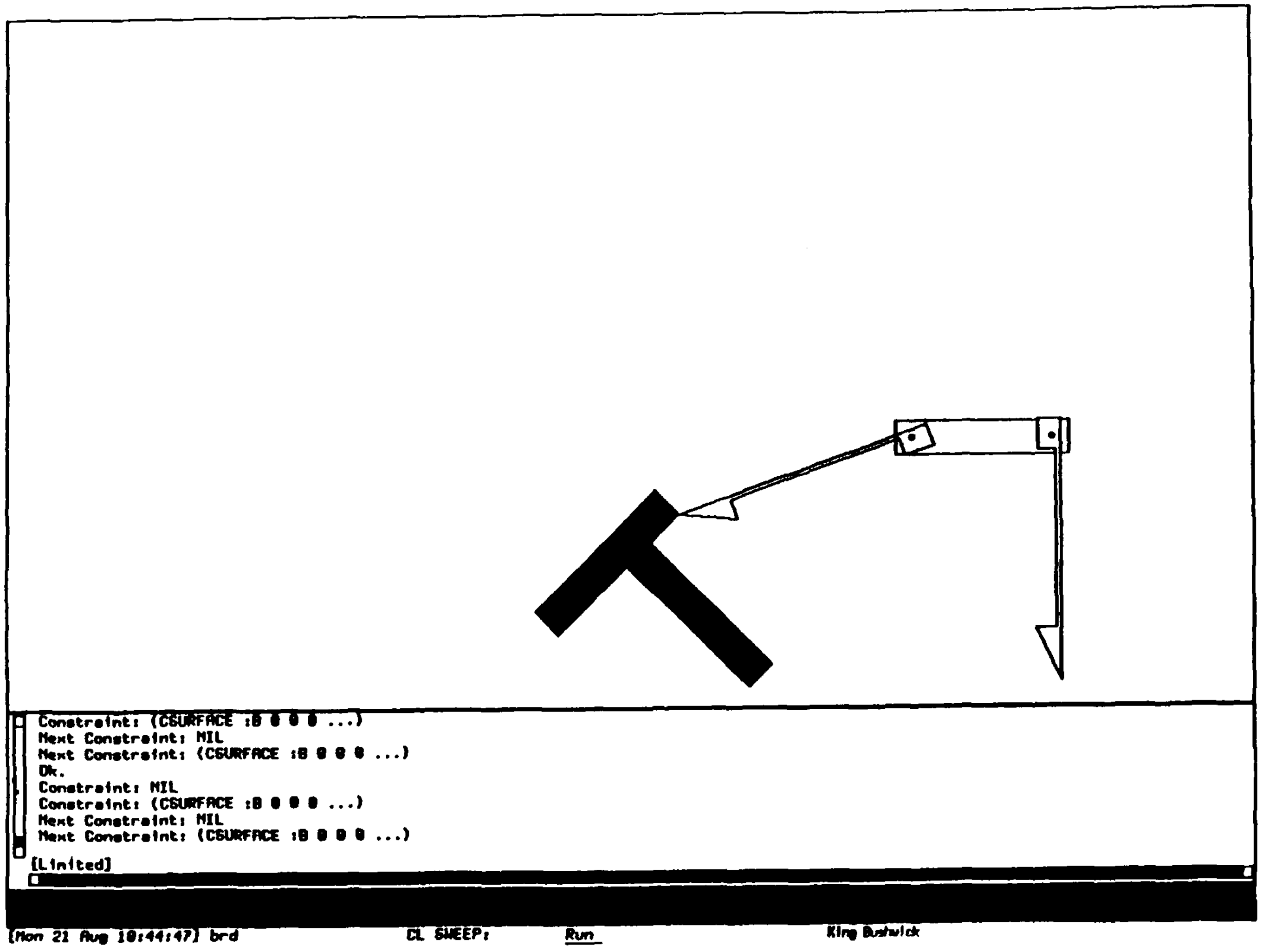
fig. 6d



```
Constraint: (CSURFACE :A -8865 -4137 4137 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)

[Limited]
```

```
[Mon 21 Aug 10:44:14] brd                CL SWEEP:     Run                   King Bushwick
```

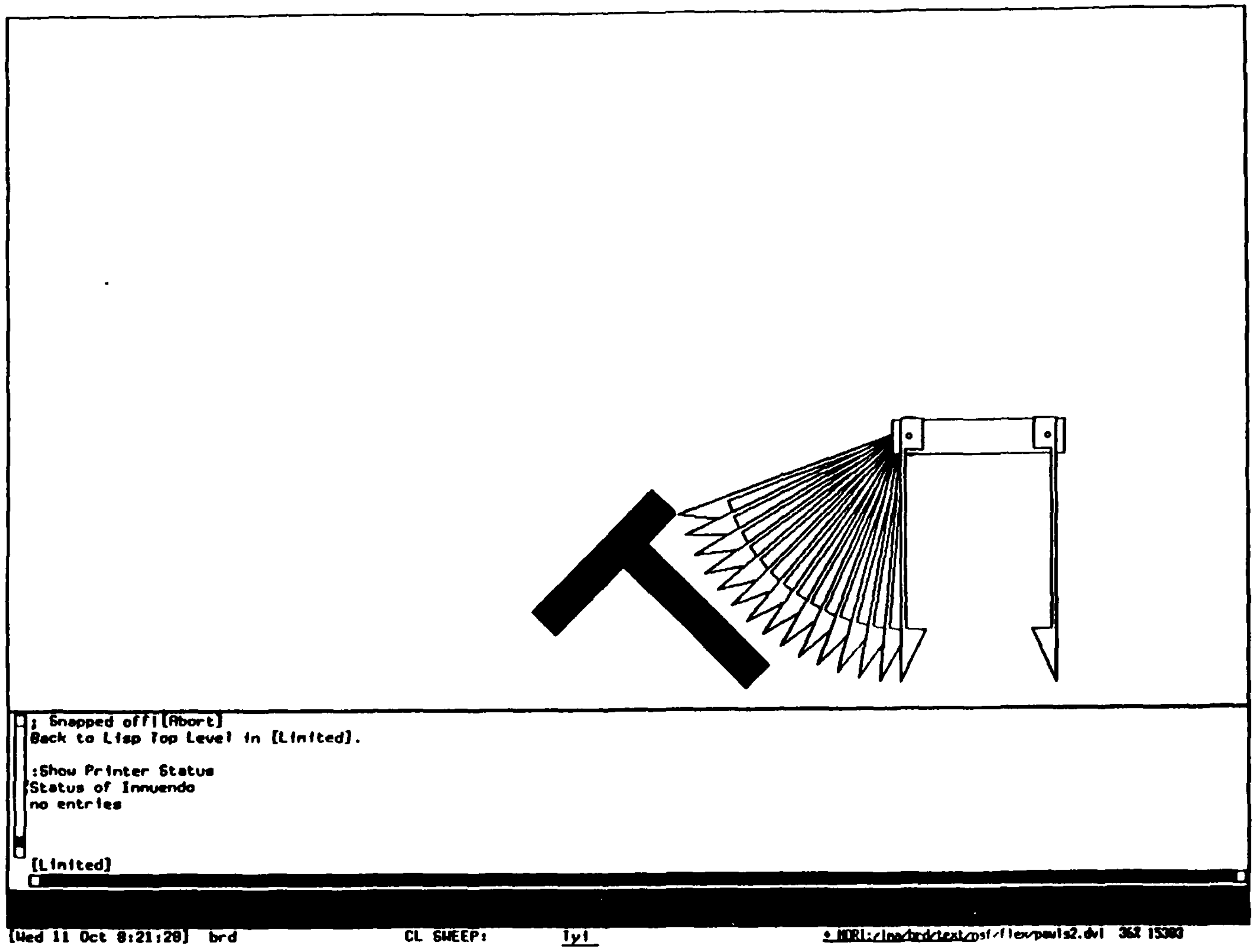
fig. 6e



```
Constraint: (CSURFACE :B 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :B 0 0 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :B 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :B 0 0 0 ...)
[Linited]
```

```
(Mon 21 Aug 10:44:47) brd          CL SWEEP:     Run          King Bushwick
```

fig. 6f



```
; Snapped off![Abort]
Back to Lisp Top Level in [Linited].

:Show Printer Status
Status of Innuendo
no entries

[Linited]
```

```
[Wed 11 Oct 8:21:28]  brd          CL SWEEP:     Tyi          • HORI:/lms/brd/text/osf/flex/pauls2.dvl  36% 15303
```

fig. 6g

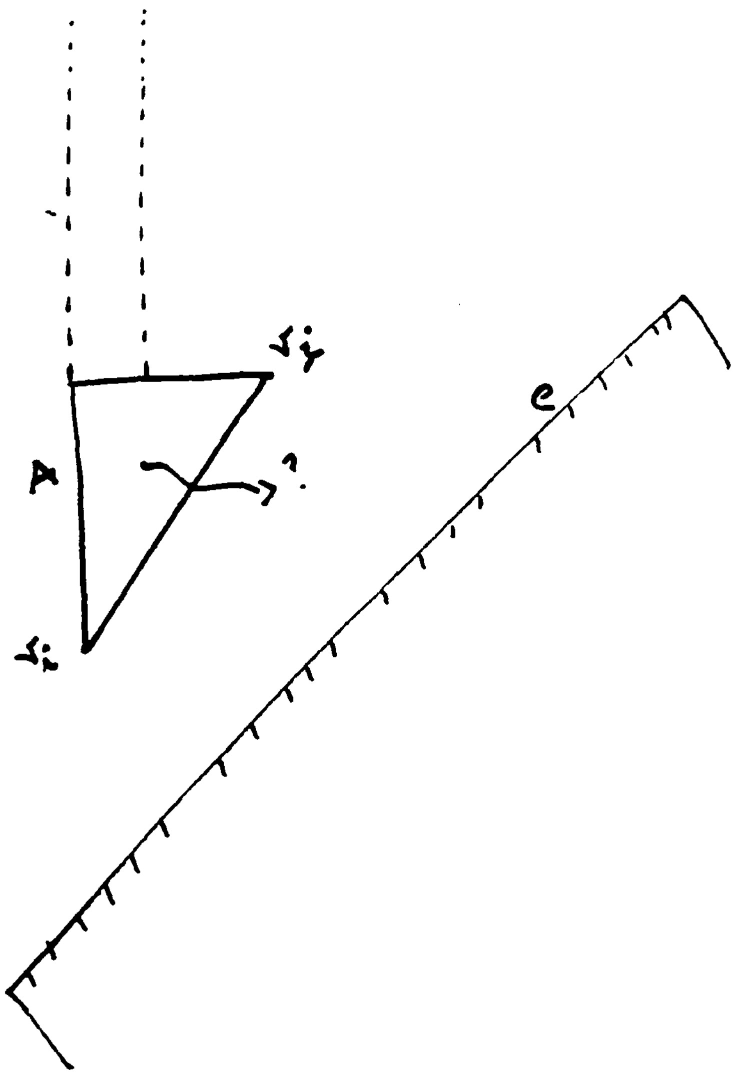


fig. 7

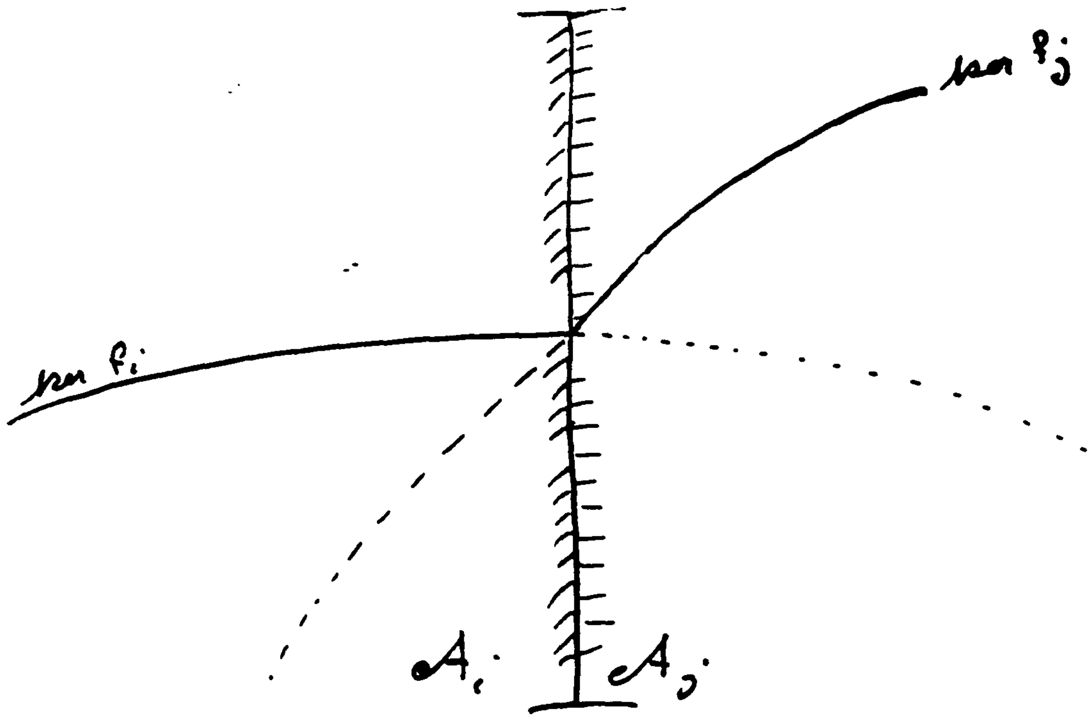


fig. 8

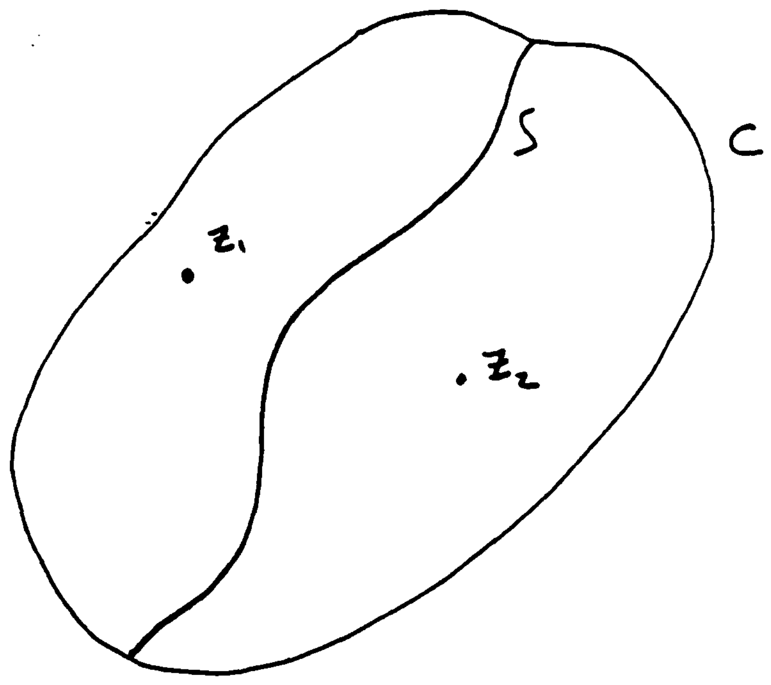


fig. 9

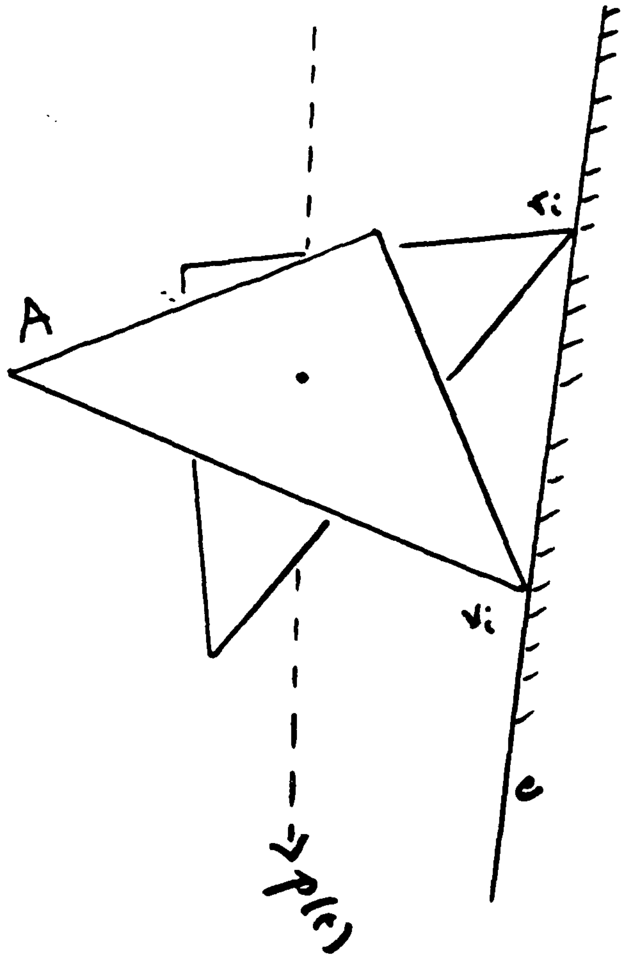


fig. 10

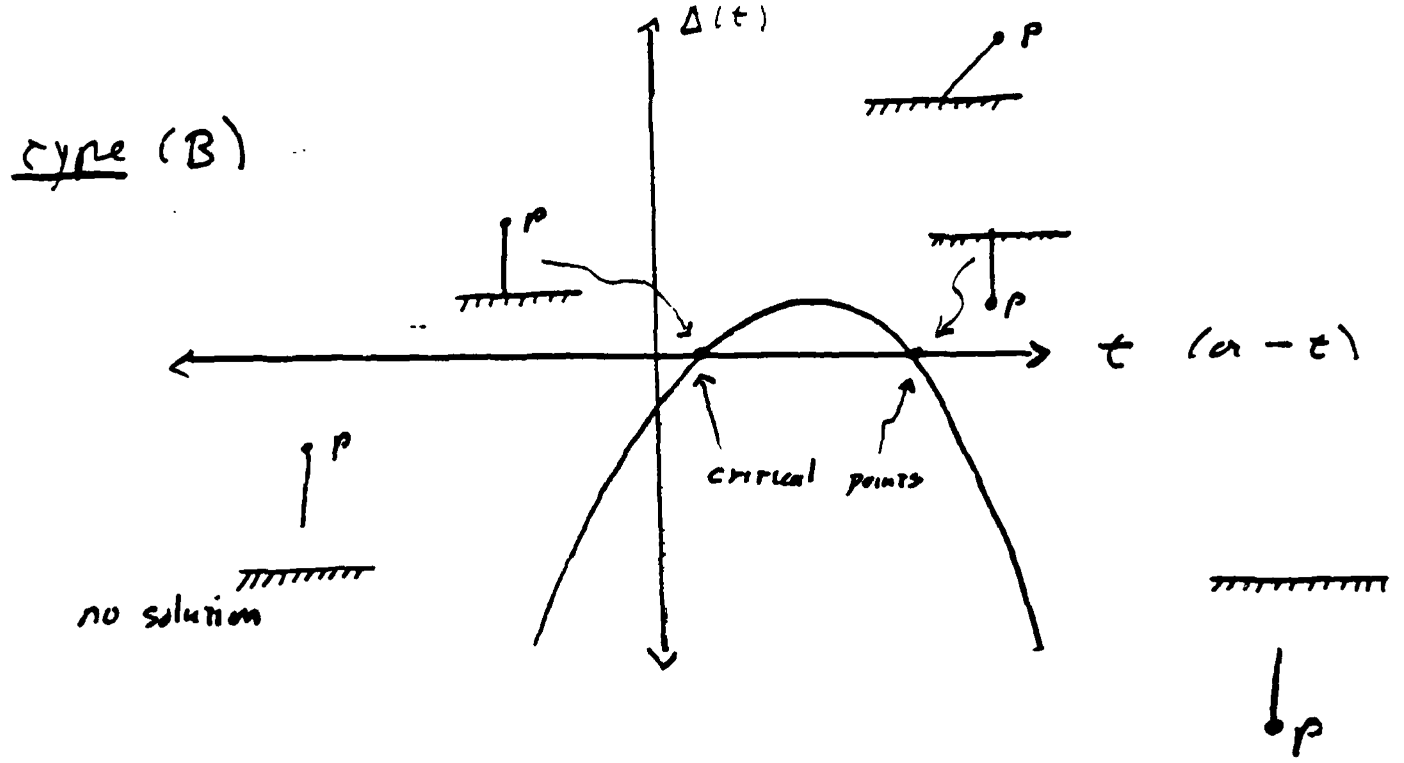
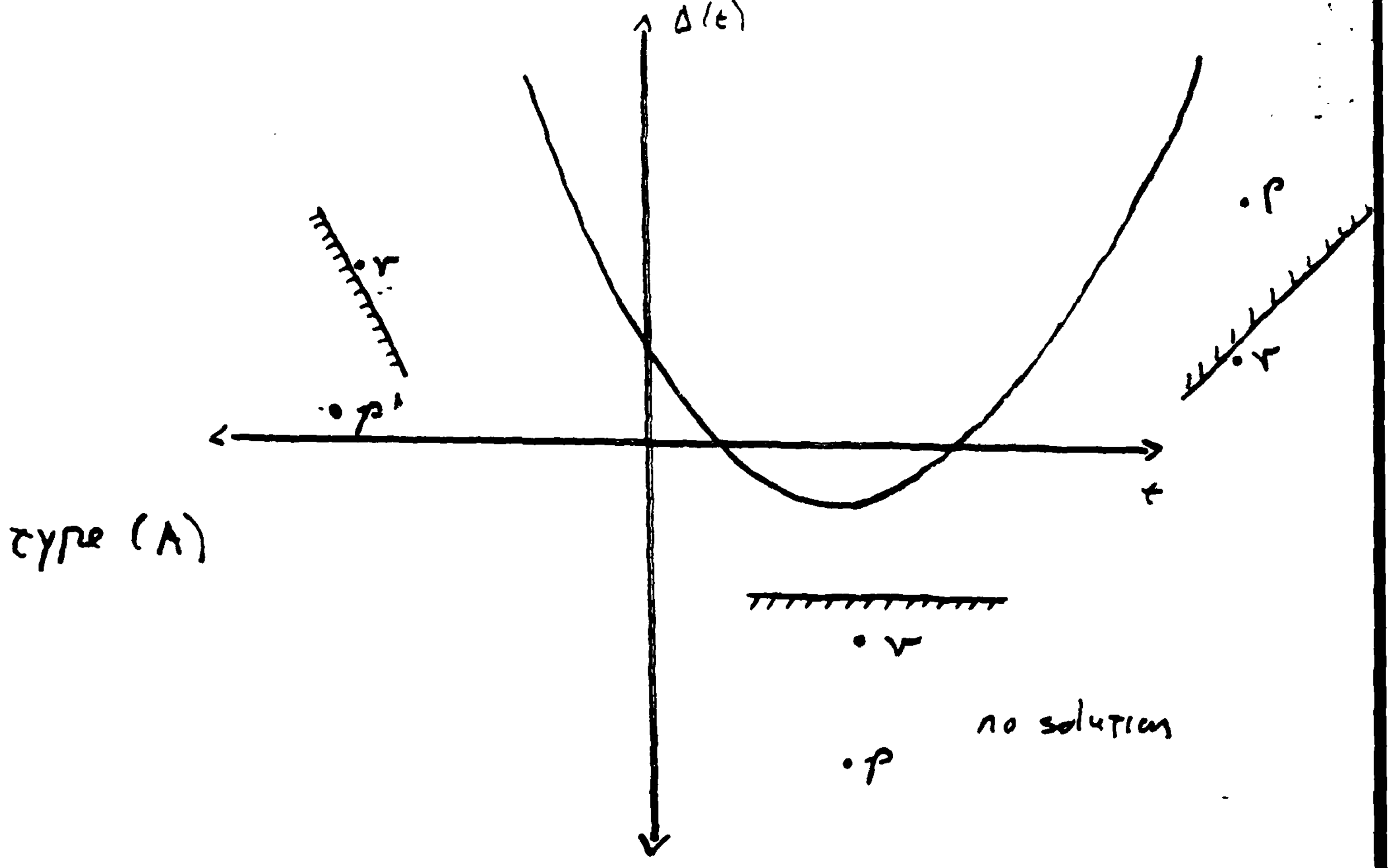


fig. 11 (type B)



Type II (type A)

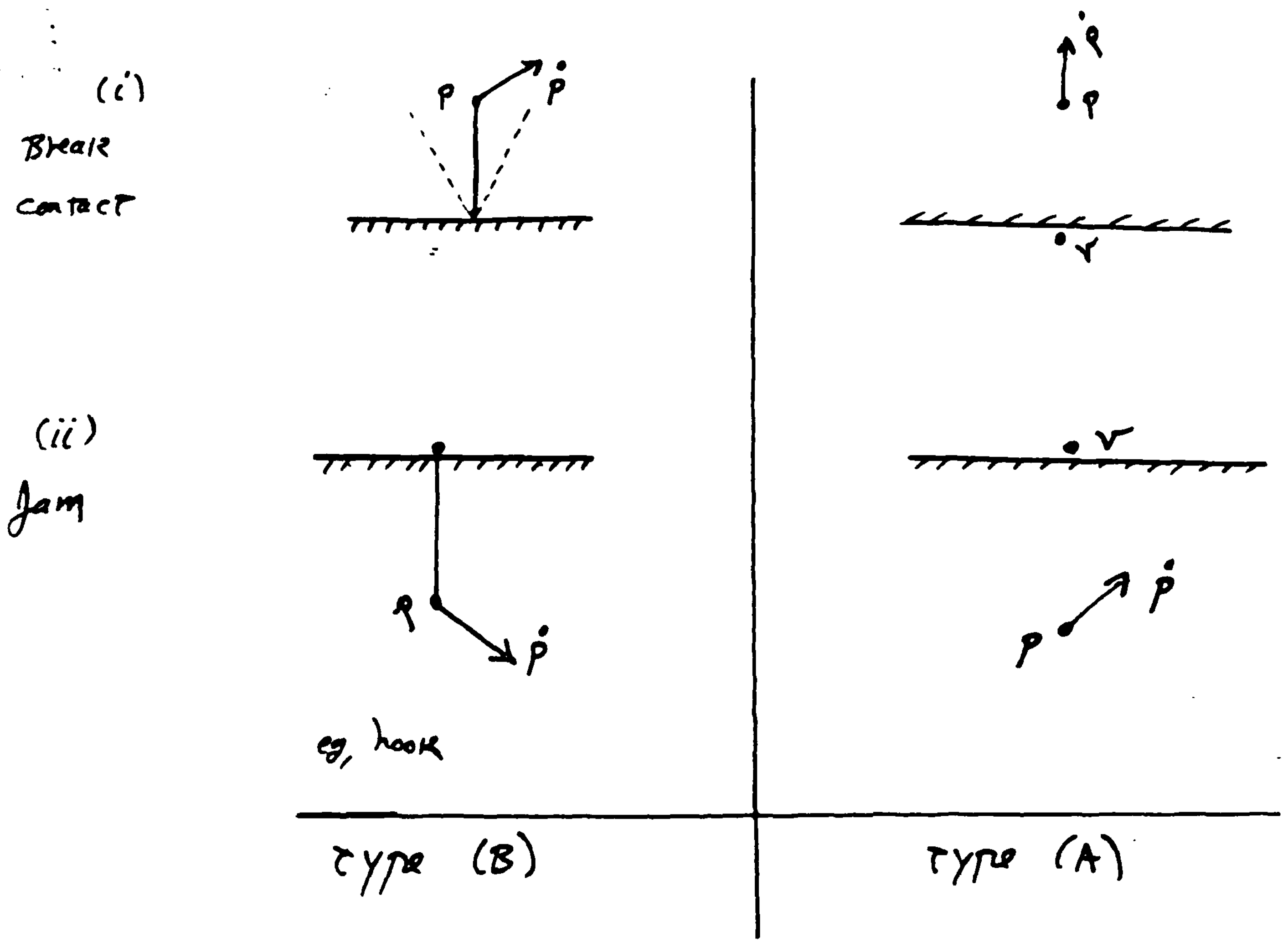
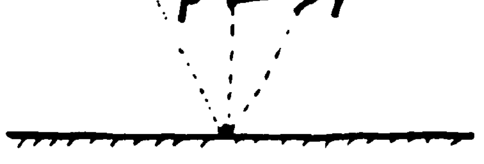


fig 12.



translation parallel to edge

fig. 13

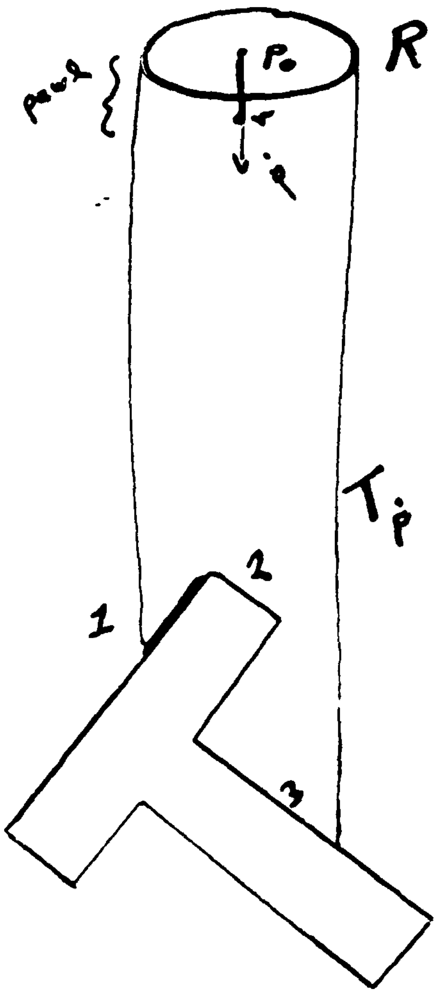


fig. 14